TUGAS AKHIR – EE 184801

PENGEMBANGAN FITUR PREDIKSI LINTASAN UNTUK PELUNCUR UAV OTOMATIS

Ahmad Fauzi Aulia
NRP 07111540000114

Dosen Pembimbing
Ronny Mardiyanto, ST., MT., Ph.D.

DEPATERMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2019

TUGAS AKHIR – EE 184801

PENGEMBANGAN FITUR PREDIKSI LINTASAN UNTUK PELUNCUR UAV OTOMATIS

Ahmad Fauzi Aulia
NRP 07111540000114

Dosen Pembimbing
Ronny Mardiyanto, ST., MT., Ph.D.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2019

FINAL PROJECT – EE 184801

DEVELOPMENT OF TRAJECTORY PREDICTION FEATURE FOR AUTOMATIC UAV LAUNCHER

Ahmad Fauzi Aulia
NRP 07111540000114

Supervisor
Ronny Mardiyanto, ST., MT., Ph.D.

ELECTRICAL ENGINEERING DEPARTMENT
Faculty of Electrical Technology
Sepuluh Nopember Institute of Technology
Surabaya 2019

PERNYATAAN KEASLIAN
TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul "Pengembangan Fitur Prediksi Lintasan untuk Peluncur UAV Otomatis" adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka. Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Surabaya, Juli 2019

Ahmad Fauzi Aulia
NRP. 0711 15 40000 114

PENGEMBANGAN FITUR PREDIKSI LINTASAN UNTUK PELUNCUR UAV OTOMATIS

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Teknik
Pada
Bidang Studi Elektronika
Departemen Teknik Elektro
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember

Menyetujui :

Dosen Pembimbing

Ronny Mardiyanto, ST., MT., Ph.D.
NIP. 198101182003121003

SURABAYA
JULI, 2019

PENGEMBANGAN FITUR PREDIKSI LINTASAN UNTUK PELUNCUR UAV OTOMATIS

Nama : Ahmad Fauzi Aulia
Pembimbing : Ronny Mardiyanto, ST., MT., Ph.D.

ABSTRAK

Prediksi lintasan UAV merupakan estimasi dari lintasan yang akan dilewati oleh UAV saat meluncur dari peluncur UAV. Saat UAV lepas landas dari peluncur UAV, maka belum tentu UAV akan langsung terbang naik dikarenakan daya angkat dari UAV belum dapat mengimbangi gaya berat. Salah satu faktor yang mempengaruhi kondisi ini adalah kecepatan. Namun, tidak semua peluncur UAV memiliki fitur yang dapat menghitung kecepatan yang diberikan sehingga sulit untuk dapat mengetahui atau mengestimasi lintasan UAV saat lepas landas. Aplikasi prediksi lintasan ini akan dapat menghitung kecepatan yang diberikan oleh peluncur UAV saat UAV lepas landas sekaligus mengestimasi lintasan terbang dari UAV tersebut, utamanya untuk dapat menghitung kapan *lift* dari UAV dapat mengimbangi *weight*, atau yang direferensikan sebagai *turning point.* Dengan menggunakan teori dasar fisika, konsep aerodinamika serta konsep gerak parabola maka dapat didapatkan posisi *x* dan *y* UAV dari setiap waktunya yang merupakan dasar dari prediksi lintasan ini. Rumus yang didapat kemudian diterjemahkan ke dalam aplikasi yang dapat menerima masukan data dan mengolahnya menjadi tampilan grafis *mapping point.* Untuk mendukung performa dari aplikasi ini, dibuat juga perangkat keras berupa *measurement box* yang dapat membantu membaca luas permukaan pesawat yang menjadi salah satu parameter dalam rumus perhitungan.

Aplikasi yang dibuat mampu melakukan prediksi lintasan atau *mapping point* lintasan terbang UAV sampai ke titik *turning point* dengan berbagai konfigurasi peluncuran dan parameter yang diberikan. Rata-rata error dari estimasi *turning point* adalah selama 0.15s dan sejauh 0.3884m, sedangkan rata-rata error dari estimasi keseluruhan lintasan adalah 0.362m.

Kata kunci: Prediksi Lintasan, *Unmanned Aerial Vehicle* (UAV), Peluncur *UAV*

DEVELOPMENT OF TRAJECTORY PREDICTION FEATURE FOR AUTOMATIC UAV LAUNCHER

Name : Ahmad Fauzi Aulia
Supervisor : Ronny Mardiyanto, ST., MT., Ph.D.

ABSTRACT

UAV trajectory prediction is a calculated estimation for trajectory that will be passed by the UAV when taking off from the launcher. On take-off, UAV will not automatically go up because the lift force has yet to compensate the weight force. One of the factor of such condition is velocity. Unfortunately, not every UAV launcher has the feature to calculate velocity given by the take-off process therefore it'll be difficult to predict or estimate UAV's trajectory when taking off. This trajectory prediction application is able to calculate velocity given by the UAV launcher as well as estimating the flight trajectory, mainly to calculate and estimate when and where the lift force is equal or greater than the weight force, or referenced as turning point. Using fundamental physic laws, the concept of aerodynamic and projectile motion, the x and y position of UAV for every index of time can be obtained, which becomes the core of this trajectory prediction. The formula then gets worked into an application which can receive data inputs and process them into a graphical result of trajectory prediction. As to further enchance the application's performance, there's also a supporting hardware called measurement box that can read the airplance's surface area which is a part of the parameter used in the calculation formula.

This trajectory prediction application is able to estimate the trajectory of UAV in the form of mapping point up until the turning point with various parameters and configurations. The average error for turning point estimation are 0.15s in time and 0.3884m in height, whereas the average error for the overall trajectory prediction is 0.362m

Keywords: Trajectory Prediction, Unmanned Aerial Vehicle (UAV), UAV Launcher

KATA PENGANTAR

Puji dan syukur penulis ucapkan kehadirat Allah SWT atas berkat dan hidayah yang diberikan, sehingga penulis dapat menyelesaikan laporan tugas akhir dengan judul "**Pengembangan Fitur Prediksi Lintasan untuk Peluncur UAV Otomatis".** Tugas akhir ini dibuat untuk memenuhi persyaratan dalam menyelesaikan pendidikan program Strata-Satu di Departemen Teknik Elektro, Fakultas Teknologi Elektro, Institut Teknologi Sepuluh Nopember.

Selama proses perancangan, pengerjaan dan penyusunan laporan, penulis mendapatkan banyak bantuan dari berbagai pihak. Untuk itu, penulis mengucapkan terima kasih kepada:

1. Ayah, ibu, dan kakak yang selalu mendukung dan membantu selama penulis mengerjakan tugas akhir.
2. Bapak Ronny Mardiyanto, ST., MT., Ph.D selaku dosen pembimbing atas gagasan topik tugas akhir serta bimbingan dan arahan untuk penulis.
3. Seluruh dosen bidang studi elektronika Departemen Teknik Elektro ITS.
4. Rekan-rekan laboratorium A206, Salik dan Wisnu dari Tim KRTI, Nida dari Matematika, dan banyak lagi yang telah banyak memberikan bantuan bagi penulis.
5. Teman-teman bidang studi elektronika yang tidak dapat disebutkan satu-persatu yang telah membantu dan memberikan semangat kepada penulis selama menjalani perkuliahan di Departemen Teknik Elektro ITS.

Penulis sadar bahwa tugas akhir ini masih jauh dari kata sempurna. Agar tugas akhir ini dapat menjadi lebih baik, penulis menerima segala kritik dan saran. Harapannya, tugas akhir ini dapat memberikan manfaat dan menginspirasi, sebagaimana penulis mendapatkan banyak manfaat dan inspirasi dari mengerjakannya.

Surabaya, Juli 2019

Penulis

DAFTAR ISI

DAFTAR GAMBAR

.....Halaman ini sengaja dikosongkan.....

DAFTAR TABEL

.....Halaman ini sengaja dikosongkan.....

BAB 1
PENDAHULUAN

Tugas akhir adalah salah satu persyaratan akademik untuk mendapat gelar sarjana teknik di Institut Teknologi Sepuluh Nopember Surabaya. Umumnya, tugas akhir berbentuk penelitian. Tugas akhir ini akan membahas tentang pembuatan aplikasi prediksi lintasan sebagai bentuk pengembangan terhadap perangkat peluncur UAV yang sudah ada.

Pada bab awal ini akan dibahas mengenai hal-hal fundamental untuk memulai sebuah penelitian, diantaranya yaitu latar belakang, perumusan masalah, tujuan, batasan masalah, metodologi, sistematika penulisan dan relevansi.

1.1. Latar Belakang

UAV merupakan bagian dari UAS atau *Unmanned Aerial System*, dimana dalam satu sistem UAS memuat beberapa bagian penting, yaitu UAV, kemudian sistem kontrol dan juga sistem *launch and retrieve*[2]. Komponen dari sistem *launch and retrieve* salah satunya adalah peluncur UAV. Peluncur UAV digunakan untuk mendukung performa UAV, utamanya pada saat penerbangan awal atau peluncuran untuk memberikan kecepatan awal. Umumnya, peluncur UAV didesain untuk dapat memberikan minimal kecepatan awal 15% lebih besar dibandingkan kecepatan *stall* dari konfigurasi UAV tersebut.[3]

Peluncur UAV memiliki beberapa konfigurasi tertentu sesuai dengan jenisnya. Semisal untuk tipe *bungee-launcher*, maka peluncur dapat diatur sedemikian rupa untuk tipe UAV yang akan diterbangkan, contohnya jumlah karet yang akan digunakan, regangan karet, dan lain sebagainya[4]. Tidak hanya peluncurnya saja, namun UAV itu sendiri juga memiliki parameter-parameter tersendiri yang berbeda dengan UAV lainnya, semisal berat, jenis motor dan propeler, serta lainnya. Konfigurasi dari peluncur UAV serta parameter-parameter dari UAV itu sendiri akan mempengaruhi hasil peluncuran[5], dan tidak mustahil apabila menggunakan konfigurasi peluncur yang kurang sesuai pada UAV yang akan diluncurkan justru akan menyebabkan UAV gagal terbang dan jatuh. Berkaitan dengan tujuan awal penggunaan peluncur, maka dapat diasumsikan bahwa salah satu faktor yang mempengaruhi kegagalan peluncuran adalah kecepatan awal yang kurang dapat

memberi dorongan pada UAV agar dapat meluncur. Sayangnya, tidak semua peluncur UAV memiliki fitur yang dapat memberikan informasi mengenai kecepatan yang akan diberikan kepada UAV dengan konfigurasi peluncur serta konfigurasi UAV yang digunakan.

Oleh karena itulah, dirancanglah sistem perhitungan yang dapat memberikan kalkulasi dari berbagai parameter, mulai dari parameter gaya dorong peluncur, daya motor UAV, percepatan dan berbagai parameter lain. Sistem baru yang dirancang ini memungkinkan bagi pengguna peluncur UAV untuk mendapatkan data berupa prediksi lintasan serta estimasi jarak ataupun lokasi pasti dari titik peluncuran dimana nantinya UAV tersebut akan mencapai titik terendahnya sebelum dapat terbang dengan optimal yang akan direferensikan sebagai *turning point*. Dengan adanya perhitungan dan tampilan nilai jarak tersebut, maka pengguna dapat memastikan konfigurasi peluncur yang digunakan sesuai dengan tipe dan konfigurasi UAV yang akan diterbangkan sehingga dapat mencegah UAV gagal terbang.

1.2. Perumusan Masalah

Perumusan masalah pada Tugas Akhir ini adalah :

1. Merumuskan perhitungan prediksi lintasan untuk UAV *Fixed Wing*.
2. Merancang sistem tampilan *software* untuk aplikasi prediksi lintasan.

1.3. Tujuan Penelitian

Penelitian pada tugas akhir ini bertujuan sebagai berikut:

1. Mendapatkan formula perhitungan untuk prediksi lintasan UAV *Fixed Wing.*
2. Membuat implementasi sistem prediksi lintasan pada UAV *Launcher* dalam bentuk *software*.

1.4. Batasan Masalah

Batasan masalah dari tugas akhir ini adalah sebagai berikut :

1. Program dibuat khusus untuk satu peluncur UAV.
2. Tidak ikut serta dalam pembuatan UAV.
3. *Lift Coefficient* dari formula kalkulasi dianggap 0.5 untuk semua jenis UAV.[6][7]

1.5. Metodologi Penelitian

Alur dari pengerjaan tugas akhir ini akan dibagi menjadi beberapa langkah, yaitu sebagai berikut :

1. Studi Literatur

Studi literatur bertujuan untuk mempelajari dan mengkaji teori dasar dan ataupun data-data penelitian sebelumnya yang relevan dengan tugas akhir yang akan dikerjakan. Literatur ini dapat diambil dari jurnal-jurnal, buku, artikel, ataupun *paper* yang memiliki keabsahan sehingga dapat dijadikan referensi. Beberapa topik yang nantinya perlu dipelajari lebih lanjut adalah ilmu dasar fisika mengenai gerak lurus berubah beraturan, momentum, resultan daya, aerodinamika, dan lain sebagainya.

2. Pengambilan Data dan Perancangan Formula Kalkulasi

Setelah mempelajari teori dasar dan melakukan tinjauan pustaka dari penelitian-penelitian sejenis, maka dilakukanlah perancangan dari formula kalkulasi yang akan digunakan sesuai dengan kondisi dari peluncur *UAV* yang ada. Formula kalkulasi ini nantinya akan melibatkan banyak variabel, baik variabel tetap maupun variabel bebas. Untuk menunjang pemahaman, dilakukanlah pula pengamatan mengenai eksekusi langsung dari alat peluncur *UAV* dan pengambilan data dari berbagai percobaan peluncuran *UAV* dengan data berupa jarak titik terendah atau titik jatuh serta gaya yang digunakan dari tiap percobaan. Data ini nantinya akan dijadikan referensi untuk pembuatan formula kalkulasi, sekaligus sebagai bahan untuk memeriksa apakah hasil akhir dari formula yang dibuat sudah sesuai atau belum.

3. Pembuatan Program dan Tampilan

Tahap ini akan berfokus ke pembuatan program serta tampilan tatap muka. Formula kalkulasi yang telah didapatkan dari langkah sebelumnya kemudian akan diterjemahkan ke dalam bentuk program. Secara garis besar, program ini akan menerima beberapa variabel masukan yang kemudian menjadi parameter utama dalam formula kalkulasi, sehingga dengan masukan yang berbeda akan menghasilkan keluaran yang berbeda pula. Sejalan dengan pembuatan algoritma program, dirancang pula tampilan tatap muka yang mampu mendukung keseluruhan *software* sehingga pengguna

dapat dengan mudah memasukkan data yang dibutuhkan, dan data yang dimasukkan dapat diakses oleh program utama agar dapat menghasilkan keluaran yang sesuai. Program ini sendiri nantinya akan dibagi menjadi 2 bagian, yaitu kalkulasi pada peluncur dan kalkulasi setelah meluncur. Apabila kalkulasi setelah meluncur telah selesai, hasil kalkulasi akan menjadi parameter untuk kalkulasi kedua yaitu kalkulasi setelah meluncur. Pada kalkulasi kedua ini, dirancang pula tampilan yang dapat memberikan gambaran mengenai prediksi lintasan dari *UAV* setelah lepas landas dari peluncur, apakah *UAV* akan jatuh atau dapat sukses terbang.

4. Pengujian Sistem

Setelah program dan tampilan tatap muka selesai dibuat, maka langkah selanjutnya adalah menguji coba dari keseluruhan program atau sistem yang dibuat. Program akan dijalankan sebelum peluncuran *UAV* untuk mendapatkan prediksi lintasan, kemudian akan dicocokkan saat *UAV* benar-benar meluncur, apakah lintasan yang ada pada *software* sesuai atau tidak. Pengujian juga akan menggunakan lebih dari satu jenis *UAV*, dengan harapan bahwa program yang dibuat dapat digunakan untuk segala jenis *UAV* dengan data-data masukan yang berbeda. Apabila dalam pengujian didapatkan hasil dengan perbedaan yang signifikan, maka perlu dilakukan analisis ulang serta evaluasi keseluruhan program, utamanya dalam formula kalkulasi. Apabila pada pengujian sistem tidak ada lagi yang perlu dievaluasi dan hasilnya sudah mendekati sesuai, maka sistem bisa dikatakan berhasil.

5. Penyusunan Laporan Tugas Akhir

Penyusunan laporan dilakukan secara paralel bersamaan dengan tahap-tahap sebelumnya, sehingga apabila ada evaluasi-evaluasi pada langkah-langkah yang ada, maka laporan bisa langsung menyesuaikan. Laporan dapat dinyatakan final apabila sistem sudah berhasil sehingga tidak perlu adanya perubahan lagi.

1.6. Sistematika Penulisan

Dalam penulisan hasil penelitian, buku Tugas Akhir ini akan dipaparkan dalam beberapa bab, sebagai berikut :

- **BAB 1 : Pendahuluan**
  Bab pertama akan memaparkan secara umum mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi, sistematika penulisan, dan relevansi.
- **BAB 2 : Tinjauan Pustaka**
  Bab ini akan membahas mengenai teori-teori yang menjadi dasar dari penyusunan laporan, umumnya mengenai komponen ataupun software yang digunakan, serta mempelajari tentang jurnal ataupun pustaka yang sudah ada yang berhubungan erat dengan tugas akhir ini.
- **BAB 3 : Perancangan Sistem**
  Bab ini menjelaskan tentang perencanaan sistem secara keseluruhan, dimulai dari perumusan formula hingga nantinya perancangan perangkat lunak (*software)* dan juga perangkat keras *(hardware)* untuk mendukung performansi dari perangkat lunak yang dirancang.
- **BAB 4 : Pengujian dan Analisis**
  Pada bab ini, hasil pengujian dari sistem yang dirancang akan dipaparkan beserta analisis mengenai keakuratan sistem ataupun hal-hal lain yang berhubungan dengan hasil sistem yang dibuat.
- **BAB 5 : Penutup**
  Bab terakhir ini akan membahas tentang kesimpulan dari sistem yang telah dirancang setelah dianalisis dan memaparkan bagian-bagian yang bisa diperbaiki atau dikembangkan lebih lanjut.

1.7. Relevansi

Adanya prediksi lintasan pada proses peluncuran UAV ini menjadi salah satu fitur yang dapat dimanfaatkan untuk mencegah terjadinya gagal terbang dari UAV, sehingga mengurangi resiko dari kerusakan UAV. Dengan adanya gambaran mengenai lintasan yang akan dilalui oleh UAV, pengguna dapat menyesuaikan kondisi lingkungan dengan kondisi ideal yang diperhitungkan oleh program sehingga UAV dapat terbang dengan baik.

.....Halaman ini sengaja dikosongkan.....

BAB 2
TINJAUAN PUSTAKA

Sebelum memulai atau mengadakan sebuah penelitian, seyogyanya dilakukan studi literatur terlebih dahulu terhadap penelitian-penelitian yang sudah ada dan sejalan dengan penelitian yang akan dilakukan maupun mempelajari landasan teori yang berkaitan erat dengan penelitian.

Bab ini akan membahas beberapa poin penting yang menjadi titik tumpu utama dari tugas akhir ini, baik dari segi hukum fisika maupun perangkat lunak dan perangkat keras yang digunakan. Untuk mendukung pemahaman pula, dilakukan tinjauan terhadap penelitian-penelitian yang menyerupai tugas akhir ini.

2.1. *Unmanned Aerial Vehicle* (UAV)

Unmanned Aerial Vehicle secara bahasa dapat diartikan sebagai kendaraan udara tanpa awak. Perangkat yang sering dikenal dengan singkatan UAV merupakan bagian dari *Unmanned Aircraft System* , bersamaan dengan perangkat peluncur dan pendaratan, sistem komunikasi antar pengguna dan UAV, perangkat kendali atau *remote control*, dan hal-hal lain yang dianggap perlu[2]. UAV sering dimanfaatkan untuk pemetaan dan pengawasan daerah atau lokasi tertentu dari ketinggian tertentu sehingga cakupan wilayah pengawasannya sangat luas dibandingkan dengan pengawasan dengan mengandalkan mobilitas manusia.

UAV pada dasarnya memiliki dua tipe, yaitu tipe *rotary-wing* dan *fixed-wing*. Tipe pertama yaitu *rotary-wing* mengandalkan rotor dan baling-baling sebagai penggerak utama. Ada beberapa tipe rotor berdasarkan jumlahnya, yaitu 4 rotor (*quadcopter*), 6 rotor (*hexacopter*), dan 8 rotor (*octacopter*). Dikarenakan *UAV* tipe ini didukung oleh motor yang lebih dari 1 dan bentuk fisik yang cukup sederhana, *rotary-wing* dapat dengan mudah terbang tanpa perlu adanya dorongan awal atau menggunakan peluncur. Kekurangan dari tipe ini adalah meskipun bentuknya mempermudah penerbangan di awal, saat berada di udara biasanya *rotary-wing* akan cenderung tidak stabil.

Gambar 2.1. UAV tipe *fixed-wing*.

Tipe kedua dari UAV adalah tipe *fixed-wing*. UAV jenis ini bentuknya menyerupai pesawat pada umumnya namun kebanyakan tidak dilengkapi dengan roda. Umumnya, UAV tipe *fixed-wing* didukung oleh motor dan baling-baling yang diletakkan baik pada bagian atas ataupun belakang, tergantung jenis dari *fixed-wing* yang dimaksudkan. Dikarenakan tidak adanya sistem dari *fixed-wing* yang mendukung peluncuran awal, kebanyakan *fixed-wing* membutukan peluncur sebagai dorongan awal agar dapat terbang. Saat sudah lepas landas dan berada di udara, tipe *fixed-wing* memiliki performa yang lebih stabil dibanding *rotary-wing* dikarenakan bentuk fisik yang mendukung teorema aerodinamika.

2.2. UAV *Launcher*

UAV *Launcher* merupakan bagian dari *Unmanned Aircraft System* sebagai *launch equipment*, dimana perangkat ini berfungsi untuk meluncurkan UAV agar dapat lepas landas dan terbang [2]. Pada bobot tertentu, UAV bisa saja lepas landas hanya dengan mengandalkan kekuatan lemparan dari manusia. Hanya saja, kebanyakan UAV utamanya jenis *fixed-wing* membutuhkan energi atau daya yang cukup besar sebagai daya dorong tambahan sebelum dapat terbang menggunakan motor sendiri. Tujuan utama dari adanya UAV *Launcher* ini adalah saat lepas landas dari peluncur, UAV memiliki kecepatan awal setidaknya 15% lebih besar dibandingkan kecepatan *stall* sesuai jenis dan bentuk UAV [2].

Gambar 2.2. Peluncur UAV tipe *bungee-launcher*[1]

Beberapa parameter penting dari peluncur UAV adalah sebagai berikut [8]:

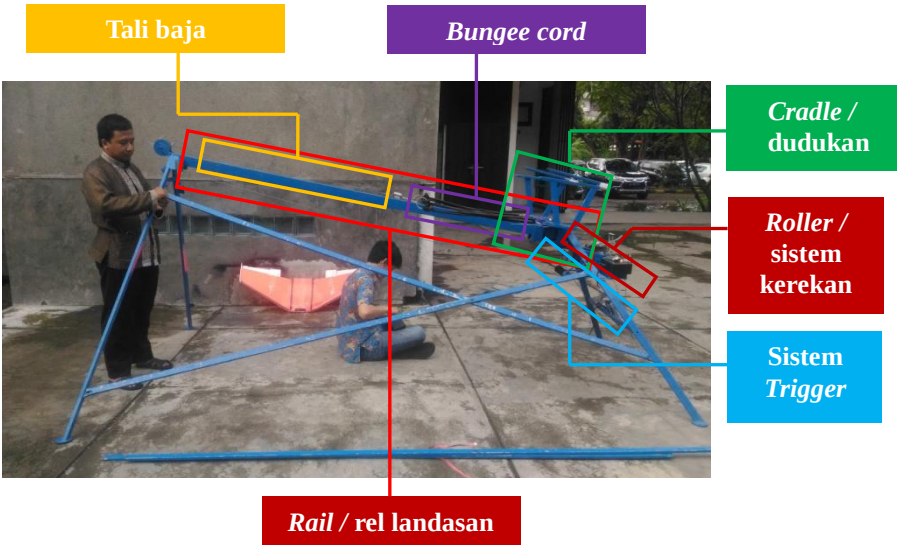
a. Kecepatan akhir peluncuran
b. Berat maksimum UAV
c. Sudut peluncuran
d. Berat peluncur
e. Panjang maksimal peluncur

Ada beberapa jenis peluncur UAV, namun secara umum dapat dikategorikan dalam 2 jenis, yaitu *military grade* UAV *Launcher* dan *light weight* UAV *Launcher*[1]. Peluncur-peluncur ini membutuhkan sumber energi yang dimanfaatkan untuk mengakselerasi UAV, olehkarenanya kebanyakan dari jenis peluncur menggunakan konsep pneumatik, per, konsep *bungee* ataupun hidraulik. Jenis *military grade* dapat mengkombinasikan dari beberapa metode di atas. Prinsip kerja dari peluncur secara garis besar adalah memiliki sistem pengunci posisi yang sederhana dari UAV dan saat kunci dilepas, maka pelontar akan meluncurkan UAV sehingga saat mencapai ujung dari peluncur maka UAV bisa meluncur dan lepas landas.

Salah satu sistem peluncur yang umum dan mudah digunakan serta dipahami adalah konsep dari peluncur tipe *bungee*. Peluncur ini memiliki karakteristik khusus yang memanfaatkan energi yang dihasilkan dari regangan karet *bungee cord* yang elastic untuk menerbangkan UAV. Gambaran umum dari peluncur ini adalah adanya

rel peluncuran yang umumnya terbuat dari logam yang diposisikan dengan sudut tertentu sesuai dengan desainer dari peluncur tersebut. Kemudian UAV yang akan diluncurkan diletakkan pada suatu dudukan yang diposisikan pada ujung bawah rel tersebut, dimana dudukan ini telah dikunci posisinya dan ditarik oleh karet. Karet ini akan dihubungkan dengan kawat atau tali baja yang mana tali baja ini akan dilewatkan pada suatu kerekan sehingga nantinya tali baja ini akan menarik karet hingga pada ujung rel landasan. Dengan menarik pengunci dudukan, maka dudukan yang terhubung dengan karet yang sudah meregang karena ditarik oleh tali baja akan terdorong ke depan hingga ke ujung landasan, dan UAV akan terakselerasi dan memiliki kecepatan awal saat meluncur dari peluncur UAV.

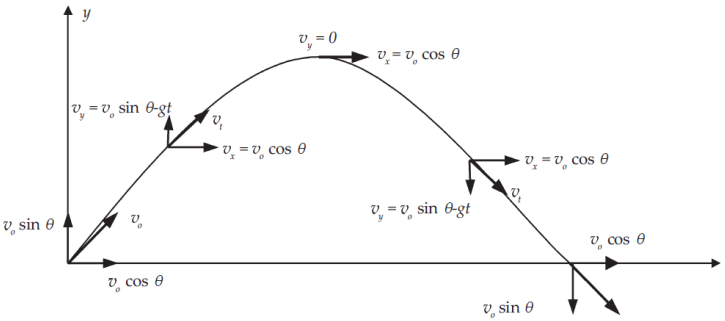
Beberapa bagian penting dari *bungee launcher* yang telah dijelaskan sebelumnya dapat dilihat pada gambar 2.3.



Gambar 2.3. Pratinjau dari komponen dalam *bunge*-launcher

2.3. Gerak Parabola

Gerak parabola merupakan salah satu gerak di bidang datar dimana lintasan dari benda yang bergerak menyerupai parabola.Pada gerak parabola, terdapat parameter sudut yaitu besar sudut awal peluncuran atau pergerakan.



Gambar 2.4. Gerak Parabola[9]

Pada gerak parabola, dengan menguraikan vektor kecepatan pada setiap waktunya pada sumbu *x* dan *y*, maka bisa dilihat bahwa benda mengalami percepatan ke bawah yaitu percepatan dari gravitasi.[9]

Untuk kecepatan pada setiap sumbunya dapat dilihat pada rumus berikut :

$$v_x = v_0 \cos \theta$$

$$v_y = v_0 \sin \theta - gt$$

Sedangkan untuk posisi setiap waktunya :

$$x = x_0 + v_0 \cos \theta \, t$$

$$y = y_0 + v_0 \sin \theta \, t - \tfrac{1}{2} gt^2$$

2.4. Hukum Hooke

Pada benda lentur, contohnya pegas, apabila pegas digantung pada satu objek dan pada ujung satunya diberikan beban *m*, maka pegas akan mengalami peregangan sepanjang *x*. Gaya ini merupakan gaya pemulih karena cenderung memulihkan atau mengembalikan benda tersebut pada posisi awalnya. Besarnya gaya yang dilakukan dapat dinyatakan dalam Hukum Hooke.

$$F_x = k \, . \, \Delta x$$

k merupakan suatu konstanta atau koefisien yang menyatakan elastisitas dari benda atau pegas tersebut.[9] Setiap benda memiliki koefisien atau konstanta yang berbeda, dan ini akan mempengaruhi benda tersebut akan kemampuannya untuk dapat meregang atau tidak.

Dari Hukum Hooke ini, dapat disimpulkan bahwa perpanjangan dari suatu benda lentur sangat berpengaruh dari gaya yang diberikan pada benda tersebut. Semakin besar gaya, maka benda akan semakin memanjang.

2.5. Hukum Newton

Pada prinsip ilmu fisika, dikenal sebutan Hukum Newton. Setidaknya ada 3 Hukum Newton yang dikenal secara umum, yaitu Hukum I Newton, Hukum II Newton, dan Hukum III Newton.

Hukum I Newton sering juga disebut dengan hukum kelembaman, dimana pada Hukum I Newton dijelaskan bahwa setiap benda akan selalu mempertahankan posisi semula. Bunyi dari Hukum I Newton adalah sebagai berikut :

"*Sebuah benda tetap pada keadaan awalnya yang diam atau bergerak dengan kecepatan konstan, jika tidak ada suatu gaya eksternal yang mempengaruhi benda tersebut."*[10]

Dengan kata lain, jika $\sum F = 0$, maka benda diam.

Hukum II Newton membahas tentang kecepatan dan percepatan. Hukum II Newton berbunyi :

"Percepatan sebuah benda berbanding terbalik dengan massa dan sebanding dengan gaya eksternal yang bekerja pada benda tersebut."

Dalam perhitungan matematis, maka dapat dituliskan :

$$F = m / a$$

Selain itu, ada juga Hukum III Newton. Hukum III Newton menjelaskan tentang prinsip aksi dan reaksi terhadap suatu objek.

$$F_{aksi} = - F_{reaksi}$$

2.6. Aerodinamika

Aerodinamika merupakan teori yang mempelari tentang dinamika aliran udara. Konsep aerodinamika ini seringkali dihubungkan dengan segala hal yang berhubungan dengan pesawat. Meskipun memang itu benar, namun sebenarnya konsep aerodinamika ini sangatlah luas cakupannya.

Beberapa frasa yang cukup sering disebutkan dalam mempelajari aerodinamika antara lain adalah *lift, weight, drag,* dan *thrust*. Keempat komponen ini sebenarnya saling berpasangan satu sama lain. Dalam pengertian yang lebih mudah, *lift* dan *weight* adalah gaya yang dialami pesawat dalam sumbu vertikal, sedangkan *drag* dan *thrust* adalah gaya yang dialami pesawat dalam sumbu horizontal.

Dalam teori aerodinamika, pesawat dikatakan mampu terbang stabil di udara apabila *lift* setidaknya sama dengan *weight* [11]. *Lift* adalah gaya angkat pesawat yang memungkinkan pesawat dapat mengudara, sedangkan *weight* adalah gaya berat yang menarik pesawat ke permukaan. Apabila *weight* lebih besar, maka pesawat akan tertarik kembali ke permukaan tanah atau dengan kata lain jatuh. Sebaliknya, apabila *lift* lebih besar, maka pesawat akan terus naik.

Lift dan *weight* dapat dituliskan dalam rumus matematika sebagai berikut :

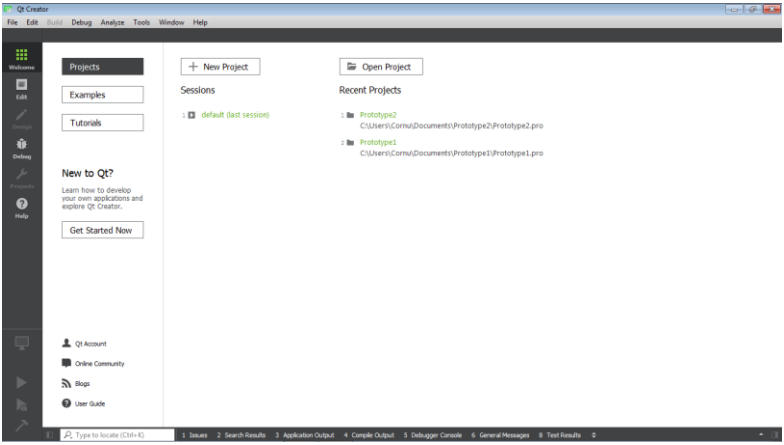
$$L = C_L \, . \, S \, . \, (½ \, . \, p \, . \, v^2)$$

$$W = m \, . \, g$$

dimana C_L adalah *coefficient of lift* yang bergantung dari tipe pesawat, *airfoil*, ketinggian, *angle of attack*, dan lain sebagainya, S adalah lebar permukaan pesawat, p adalah tekanan udara pada ketinggian saat itu, dan v adalah kecepatan pesawat. Sedangkan *weight* merupakan persamaan umum perkalian massa dengan gravitasi.

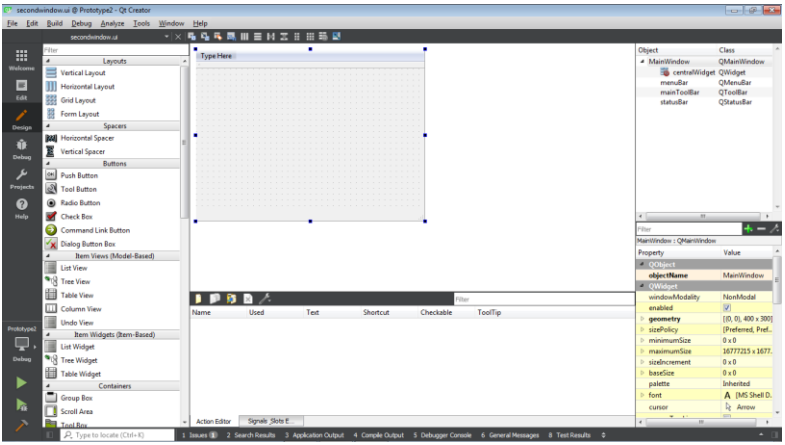
2.7. QT *Creator*

QT *Creator* merupakan *framework* untuk pengembangan program *graphic user interface* atau yang dikenal dengan GUI. Aplikasi ini berisi kumpulan kelas yang telah dirancang dan didesain untuk ‘siap-pakai’ [12] dalam menjalankan fungsi utamanya, utamanya dalam mengembangkan program visual menggunakan bahasa C++.



Gambar 2.5. *Homepage* dari QT *Creator*

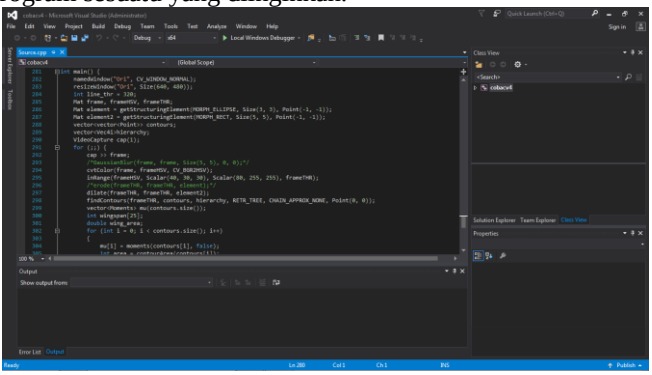
Salah satu keunggulan QT *Creator* adalah aplikasi ini telah dilengkapi dengan berbagai *widget* siap pakai yang bisa langsung digunakan untuk mendesain antarmuka dari aplikasi yang ingin dibuat, yang sering dikenal dengan sebutan QT *Designer.*



Gambar 2.6. Tampilan QT *Designer*

2.8. Microsoft Visual Studio

Microsoft Visual Studio adalah aplikasi yang dapat dimanfaatkan untuk mengembangkan program komputer atau situs web dalam bentuk aplikasi, baik dalam bentuk *console, windows*, ataupun yang lainnya. MS Visual Studio juga seringkali disebut dengan IDE atau *Integrated Development Environment*, dimana yang dimaksud IDE disini adalah perangkat yang telah dilengkapi dengan fungsi-fungsi tertentu dan khusus yang dapat dimanfaatkan dengan mudah untuk membuat atau memprogram sesuatu yang diinginkan.



Gambar 2.7. *User interface* dari MS *Visual Studio* 2015

Visual Studio memungkinkan pengguna untuk menggunakan sekitar hampir 36 bahasa pemrograman yang berbeda, baik dalam hal mengedit kode maupun saat debugging. Bahasa pemrograman yang dimaksud terdiri dari *C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML,* dan *CSS*. Bahasa pendukung lainnya seperti *Phyton, Ruby, Node.js,* dan *M* bisa digunakan dengan memasang *plug-in* terlebih dahulu.

2.9. OpenCV

OpenCV adalah salah satu librari komputer visual dan *machine learning* yang bersifat *open-source* atau terbuka untuk umum. OpenCV didesain untuk efisiensi perhitungan dan berfokus untuk aplikasi *real-time*[13]. Sesuai dengan namanya, maka OpenCV akan sangat erat berhubungan dengan penggunaan kamera secara hardware atau yang sering dikenal dengan istilah *computer vision*. *Computer vision* sendiri bisa diartikan sebagai transformasi data dari video ataupun gambar menjadi sesuatu yang lain setelah melalui proses tertentu.

OpenCV menawarkan berbagai fungsi dan perintah khusus yang dapat membantu pengguna untuk mencapai tujuan dari *computer vision* itu sendiri, mulai dari mengenali objek dari segi warna, bentuk, membaca luasan permukaan benda, membaca pergerakan, menghitung jumlah kendaraan yang lewat, hingga mengenali wajah seseorang. OpenCV sendiri paling sering ditemui diprogram menggunakan bahasa C++ dan Python. Dikarenakan sifatnya yang *open-source*, maka tidak heran jika OpenCV memiliki banyak pengguna dan komunitas.

2.10. BMP180

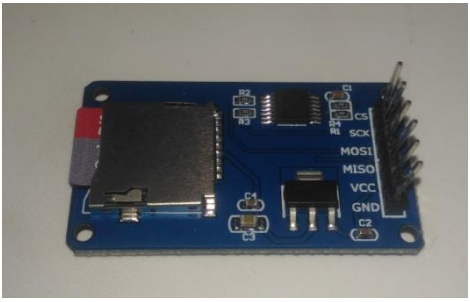
BMP180 merupakan perangkat MEMS sensor tekanan dengan presisi tinggi. Dengan penggunaan daya yang rendah, tegangan yang kecil, ukuran yang kecil, BMP180 sangat cocok digunakan pada aplikasi-aplikasi yang membutuhkan perangkat yang berukuran kecil dan tidak membutuhkan tegangan tinggi. Error pada BMP180 pun hanya berkisar pada angka 0.25m dan sensor ini pun mampu mengkonversi dalam waktu singkat [14], sehingga dapat dikatakan bahwa BMP180 sangat mumpuni untuk digunakan.

Gambar 2.8. Sensor BMP180

BMP180 didesain untuk dapat terkoneksi langsung dengan mikrokontroler melalui koneksi I2C. BMP180 dapat beroperasi pada tekanan 300hPA hingga 1100hPA dan suhu dari 0ºC hingga 85ºC, dengan tegangan operasi 3.3V. Memberikan tegangan 5V dapat beresiko merusak sensor.

2.11. *Micro* SD *Card Module*

Micro SD *Card Module* merupakan perangkat yang memungkinkan untuk membaca konten pada kartu memori ataupun membuat *file* tertentu. Dengan komunikasi SPI, maka perangkat ini sudah dapat diakses. Oleh karenanya, kebanyakan mikrokontroler yang ada seperti Arduino, NodeMCU, maupun Raspberry Pi dapat dengan mudah mengakses modul ini. [15]



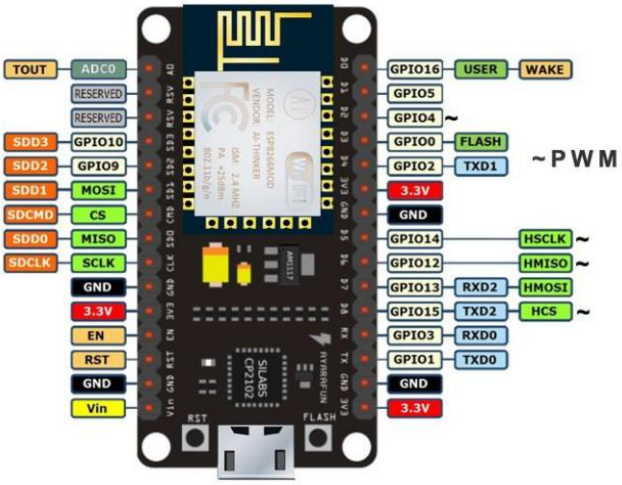
Gambar 2.9. Modul *Micro* SD *Card*

Modul ini membutuhkan tegangan 5V agar dapat aktif. Ada modul sejenis yang dapat digunakan dengan tegangan 3.3V, namun harus menggunakan adapter untuk kartu memorinya.

Perangkat ini sangat sering digunakan untuk kebutuhan *data-logging*, utamanya dikarenakan data yang didapatkan berjumlah sangat banyak sehingga akan jauh lebih efektif jika disetorkan pada kartu memori. Dengan menyetorkan data ke kartu memori, pengguna dapat mengakses data tanpa harus lewat mikrokontroler terkait.

2.12. NodeMCU

NodeMCU adalah *open-source firmware* yang dirancang untuk dapat mempermudah pengguna dalam membuat projek yang berhubungan dengan *Internet of Things*[16]. NodeMCU memiliki pin GPIO yang dapat digunakan untuk komunikasi I2C, SPI, serial maupun UART, dan dapat menghasilkan sinyal PWM maupun keluaran *high / low*. Dibandingkan perangkat seperti Arduino Nano ataupun Uno, NodeMCU dibekali *power port* dengan USB *type B*, jenis kabel yang umum digunakan sebagai *charger* HP sehingga mempermudah pengguna untuk dapat mengkoneksikan NodeMCU ke komputer.



Gambar 2.10. *Pinout* NodeMCU[16]

Poin plus lain dari NodeMCU adalah perangkat ini dapat dengan mudah diakses melalui ArduinoIDE dan dikode layaknya Arduino. Untuk dapat melakukan hal ini, cukup memasang ekstensi untuk *board* NodeMCU pada ArduinoIDE, dan NodeMCU sudah bisa diprogram layaknya Arduino.

2.13. Tinjauan Pustaka

Tinjauan pustaka yang dimaksud disini adalah membandingkan perangkat atau metode penelitian yang dirancang pada tugas akhir ini dengan perangkat atau metode penelitian serupa yang sudah ada sebelumnya. Beberapa judul paper yang menjadi tinjauan pustaka untuk tugas akhir ini akan dijelaskan di bawah.

2.13.1. *Analysis of a UAV Bungee Cord Launching Device* [5]

Penelitian ini berfokus untuk melakukan analisis mengenai performansi dari peluncur UAV dengan tipe *bungee launcher.* Pada langkah awal, dilakukan permodelan matematis dan pembuatan *free body diagram* dari peluncur UAV untuk dapat mengetahui parameter-parameter apa saja yang menjadi penentu. Kemudian dilakukan beberapa asumsi untuk menyederhanakan persamaan dan dilakukan penurunan diferensial dan memasukkan nilai awal dari peluncuran hingga didapatkan formula dari kecepatan dan waktu dari peluncur UAV terkait. Formula ini kemudian dicoba dengan menggunakan nilai *q* atau elastisitas *bungee cord* fiktif dan kemudian dibandingkan lagi dengan menggunakan nilai *q* yang riil. Hasil yang lebih akurat adalah menggunakan nilai *q* yang riil. Hasil ini kemudian dibandingkan lagi dengan percobaan yang memperhatikan faktor aerodinamis, dan ternyata tidak berbeda jauh. Kesimpulannya faktor aerodinamis bisa dihiraukan apabila berkutat pada kecepatan dibawah 30m/s.

2.13.2. *An Empirical Method for the Catapult Performance Assesment of the BPPT-developed UAVs* [17]

Performa dari peluncur ketapel akan dinilai dengan menggunakan metode empiris. Metode yang dimaksud adalah menggunakan *dummy* UAV yang dijalankan selayaknya UAV asli untuk mendapatkan data-data yang dibutuhkan. Penelitian ini mencoba mendapatkan nilai

kecepatan dari peluncur ketapel UAV dengan cara menghitung jarak jatuh dari *dummy* UAV sebagai parameter utamanya. Dengan didapatkan nilai kecepatan dari jarak jatuh, maka bisa dihitung pula kecepatan untuk UAV lainnya dengan menggunakan formula tertentu yang akan menggunakan formula dari UAV *dummy* sebagai referensinya. Hasil dari penelitian ini bisa dibilang cukup akurat dengan error hanya berkisar 4.86% saja, membuktikan bahwa untuk mendapatkan nilai kecepatan dari peluncur UAV bisa menggunakan metode ini.

2.13.3. *Increasing Launch Capability of UAV Bungee Catapult* [18]

Penelitian ini melanjutkan dari penelitian sebelumnya yang menganalisis performa dari peluncur UAV tipe *bungee launcher*. Hanya saja, kali ini digunakan tambahan *non-elastic cord* untuk mendukung performa dari *elastic bungee cord*. Dilakukan perancangan model matematis dan *free body diagram* dari peluncur baru yang didesain ini, kemudian didapatkan formula baru yang memperhitungkan *non-elastic cord*. Kesimpulannya, menggunakan peluncur model ini memiliki efisiensi tenaga sebesar hampir 90%.

2.13.4. *Design and Analysis Performance of Fixed Wing VTOL UAV*[19]

Fokus utama dalam penelitian ini adalah memodifikasi *ready-to-fly* (RTF) *fixed wing* UAV dengan menambahkan motor pada posisi tertentu sehingga UAV ini dapat terbang dalam dua mode, yaitu FW atau *fixed wing* dan MR atau *multi-rotor*. Untuk dapat memastikan kedua mode berjalan dengan baik, maka dilakukan analisis untuk menentukan minimal *thrust* dan juga daya yang digunakan untuk kedua mode. Dalam menentukan *thrust* minimal, dilakukan analisis matematika dari segi *drag* dan *lift*, dimana didapatkan fungsi *thrust* dan kemudian fungsi daya yang merupakan fungsi dari kecepatan dan *thrust*.

2.13.5. *Design Optimization of a Fixed Wing Aircraft*[6]

UAV tipe *fixed wing* umumnya membutuhkan daerah yang luas untuk melakukan proses lepas landas maupun pendaratan, sehingga jarak yang dibutuhkan untuk proses terbang dan mendarat merupakan faktor yang cukup penting. Untuk dapat mewujudkan UAV dengan jarak proses terbang maupun mendarat yang tidak memakan banyak tempat, maka perlu memperhatikan parameter-parameter tertentu, salah satunya adalah desain dari UAV itu sendiri, baik dari segi model sayap, model

ekor, motor yang digunakan, penempatan motor, jenis propeller, jumlah propeller dan lain sebagainya. Dengan melakukan analisis dan simulasi, maka didapatkan kesimpulan bahwa salah satu faktor yang dapat menghasilkan UAV yang lebih baik adalah *angle of attack* sebesar 5º.

2.13.6. *Integration of Tactical – Medium Range UAV and Catapult Launch System*[3]

Penelitian ini menitikberatkan pada analisis terhadap peluncur UAV tipe *catapult* atau ketapel yang ada di pasaran untuk UAV yang dimiliki pasukan Serbian, yaitu UAV tipe *medium-range*. Agar UAV yang dimaksud dapat diterbangkan melalui peluncur tipe ketapel, maka dilakukan analisis mengenai model UAV tersebut untuk dapat menentukan titik atau posisi yang terbaik dari UAV saat ditempatkan di atas dudukan dari peluncur. Peluncur UAV tipe ketapel yang digunakan juga harus memenuhi kualifikasi agar dapat menerbangkan UAV tipe *medium range* itu.

2.13.7. Rancang Bangun *Bungee Cord Unmanned Aerial Vehicle* (UAV) *Launcher* Otomatis[4]

Pada satu sistem UAS atau *Unmanned Aerial System*, salah satu bagian penting dalam sistem tersebut adalah peluncur. Oleh karenanya, penelitian ini akan berfokus untuk membuat peluncur yang berbasis *bungee cord* atau yang dikenal dengan *bungee catapult*. Konsep kerja dari peluncur ini adalah UAV yang akan diterbangkan diletakkan pada suatu dudukan dimana dudukan ini akan ditahan oleh sistem *trigger*. Selama dudukan tertahan, karet atau *bungee cord* akan menarik dudukan sampai pada nilai *tension* tertentu, sehingga saat sistem *trigger* dilepas karet akan menarik dudukan dan meluncurkan UAV. Peluncur UAV ini dibuat dengan beberapa sensor, yaitu sistem TX RX untuk komunikasi antara remot kontrol dengan peluncur, sensor MPU6050 untuk membaca sudut, kemudian sensor *loadcell* untuk membaca *tension* pada dudukan, serta motor DC sebagai komponen utama untuk menarik dan menegangkan karet.

2.13.8. *Study on Object Detection using OpenCV – Python*[20]

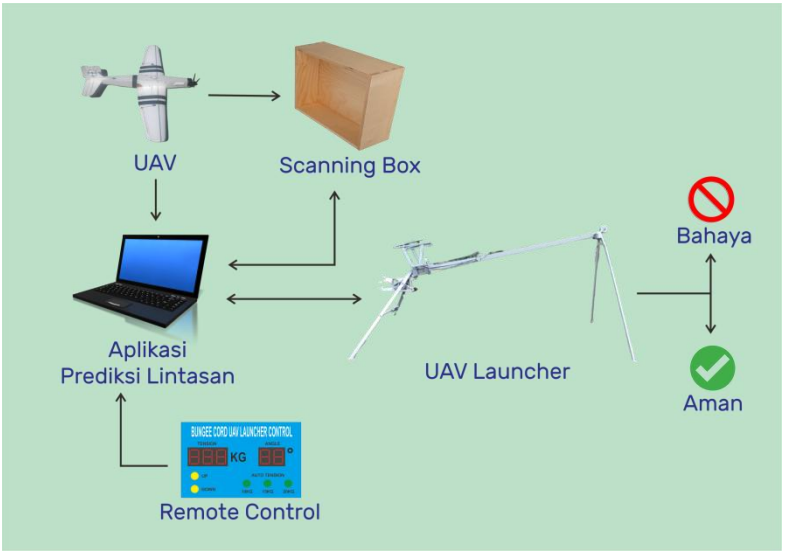
OpenCV merupakan *library* yang didesain khusus untuk melakukan pengolahan gambar maupun video. Dengan memanfaatkan *library* ini,

maka pengguna akan mendapatkan kemudahan untuk dapat mengolah gambar maupun video sesuai keinginan. Beberapa jenis implementasi deteksi objek pada OpenCV bahasa Python diantaranya adalah *Haar-like feature*, *Circular Hough Transformation*, *Template Matching*, *Blob Detection*, *Gradient-base Method*, *Local Binary Pattern*, dan lain-lain. Aplikasinya pada dunia nyata antara lain deteksi wajah, menghitung jumlah kendaraan ataupun orang, deteksi kendaraan, dan penerapan di bidang industri, keamanan, medis, maupun *human-machine interface* (HMI).

2.13.9. *Analysis Thrust for Different Kind of Propellers*[21]

Penelitian ini menguji beberapa jenis propeler dengan menggunakan motor yang sama, mengingat bahwa propeler merupakan salah satu faktor yang penting dan mempengaruhi dalam dunia desain rotor, baik yang digunakan dalam *drone*, UAV, helikopter, maupun lainnya. Hasil yang didapatkan adalah dengan menggunakan motor yang sama, *thrust* yang dihasilkan akan cenderung naik dengan semakin panjang diameter serta *pitch* dari propeler yang digunakan. Selain itu juga, propeler dengan diameter dan *pitch* yang sama namun jumlah *blade* yang berbeda ternyata tidak menghasilkan perbedaan yang signifikan terhadap *thrust*.

BAB 3
PERANCANGAN SISTEM

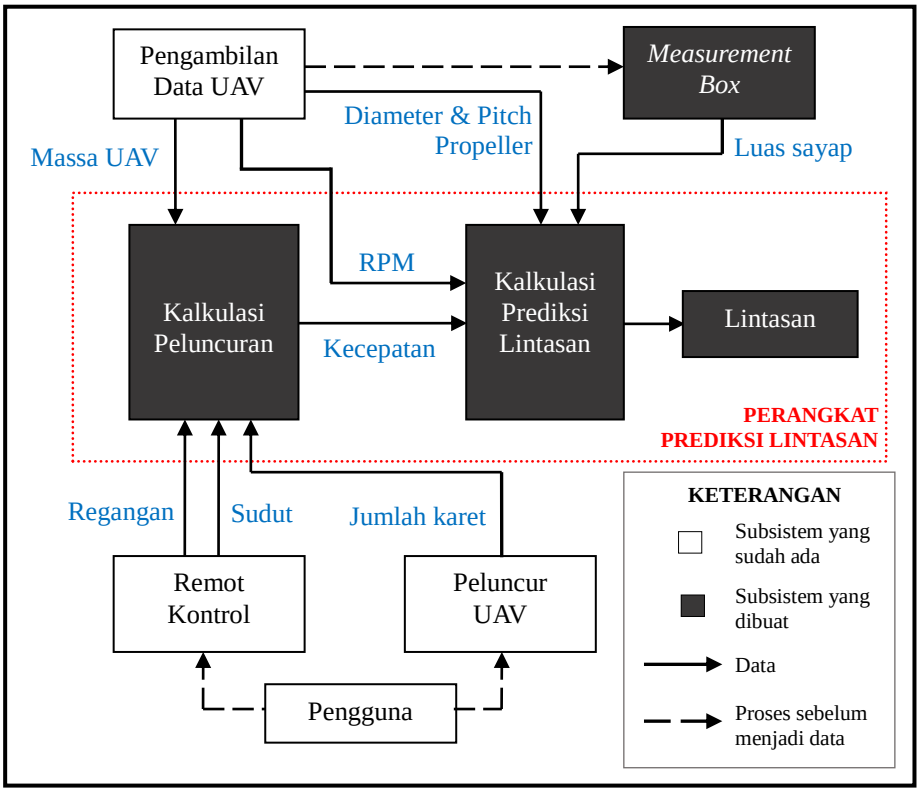


Gambar 3.1. Ilustrasi Sistem

Pada bab ini akan dipaparkan perancangan sistem secara menyeluruh. Tugas akhir ini akan berfokus kepada pembuatan aplikasi untuk menghitung dan membuat prediksi lintasan dari UAV ketika sudah lepas landas dari peluncur UAV. Sebelum dapat membuat prediksi lintasan, terlebih dahulu diperlukan perhitungan matematis dalam proses peluncuran untuk dapat mengetahui kecepatan awal dari UAV yang didapat dari proses peluncuran, yang mana peluncur UAV memang dirancang untuk memberikan dorongan awal bagi UAV agar dapat terbang. Kecepatan awal ini akan menjadi salah satu parameter penentu apakah UAV dapat terbang ataukah terjatuh. Semua hal ini akan dikemas dalam satu aplikasi yang dapat memproses data masukan dan menghasilkan keluaran berupa skema prediksi lintasan terbang UAV.

3.1. Diagram Blok Sistem

Rangkaian dan alur cara kerja sistem digambarkan dalam skema di bawah ini.



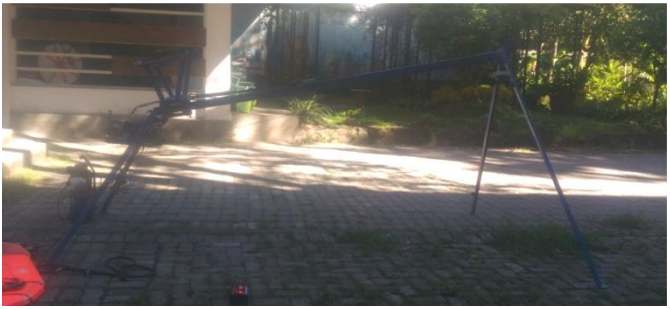
Gambar 3.2. Diagram Blok Sistem

Sistem akan membutuhkan beberapa data dari berbagai bagian subsistem. Untuk proses pertama yaitu kalkulasi peluncuran, dibutuhkan data-data berupa regangan dan sudut dari remot kontrol, kemudian jumlah karet dari peluncur UAV dimana kedua subsistem ini dikontrol oleh pengguna, serta pengambilan data UAV berupa massa. Data kemudian diproses oleh sistem dan menghasilkan data kecepatan yang akan digunakan untuk proses selanjutnya. Kemudian untuk kalkulasi

prediksi lintasan, dibutuhkan data-data berupa diameter dan *pitch* dari propeler UAV, kemudian RPM motor UAV, serta luas sayap UAV dimana data tersebu didapatkan dari proses melewati *measurement box.* Kalkulasi prediksi lintasan inilah yang akan lintasan dimana prediksi lintasan yang dimaksud adalah sejenis *mapping point* posisi *x* dan *y* dari setiap satuan waktunya.

3.2. Kalkulasi Peluncuran

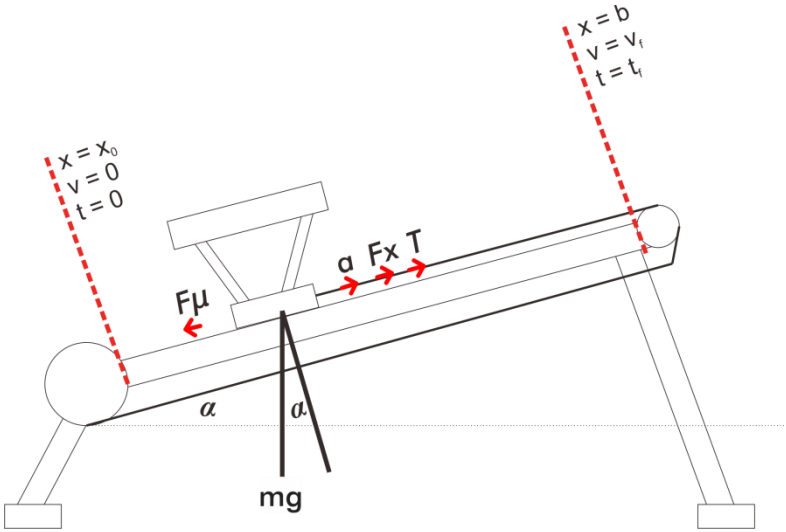
Untuk bisa mendapatkan formula kalkulasi dari peluncur UAV, maka perlu dilakukan permodelan matematis dengan asumsi-asumsi tertentu.



Gambar 3.3. Model peluncur UAV yang digunakan

Secara garis besar, perangkat peluncur UAV ini bekerja dengan cara menahan *cradle* dengan sistem trigger, kemudian dengan menggunakan pengendali pengguna akan menentukan nilai *tension. Cradle* yang tertahan akan ditarik oleh *bungee cord* dan *bungee cord* akan mengalami peregangan dengan semakin naiknya *tension.* Jadi, peluncur ini mengandalkan gaya dari regangan *bungee cord* yang akan menarik UAV sampai ke ujung peluncur dan terbang.

Dari gambaran umum tersebut kemudian diterjemahkan ke dalam bentuk *free body diagram*, yang mana merujuk kepada salah satu *paper* yang menyerupai.[5]



Gambar 3.4. *Free body diagram* peluncur UAV

Dengan melihat *free body diagram*, maka ada beberapa parameter yang menjadi penentu, utamanya dari sumbu *x*. Pada kondisi awal, maka kecepatan (*v)* dan waktu (*t)* = 0. Setelah sistem *trigger* ditarik, maka *cradle* akan bergerak ke titik ujung peluncuran, dimana kecepatan pada saat itu *(v_f)* adalah sesuai dengan waktu itu *(t_f).*

Dengan menggunakan Hukum I Newton diketahui bahwa :

$$\sum F = ma \qquad (1)$$

Gaya yang berlaku dan arahnya kemudian dimasukkan pada persamaan (1).

$$F_x - F_\mu + mg \sin \alpha = ma \qquad (2)$$

F_x adalah gaya yang dihasilkan oleh *bungee cord* yang meregang terhadap beban. Sesuai dengan Hukum Hooke :

$$F_x = q\,(x - b) \qquad (3)$$

dimana q = koefisien elastisitas, x adalah panjang karet setelah peregangan dan b panjang awal karet. Kemudian ada F_μ atau gaya gesek, dimana persamaan gaya gesek adalah :

$$F\mu = \mu \, . \, N \tag{4}$$

Karena peluncur UAV ini bukanlah bidang datar, maka ada faktor sudut yang akan mempengaruhi gaya normal atau *N*.

$$N = mg \cos \alpha \tag{5}$$

Maka dapat disimpulkan :

$$F_\mu = \mu \, . \, mg \cos \alpha \tag{6}$$

Memasukkan persamaan (3) dan (6) pada persamaan (2), maka didapatkan :

$$ma = q \, (x\text{-}b) + mg \, \mu \cos \alpha + mg \sin \alpha \tag{7}$$

a adalah turunan kedua dari *x.* Oleh karenanya, parameter *x* kemudian dikeluarkan dari kurung dan digabungkan dengan *a*.

$$ma = qx - qb + mg \, (\mu \cos \alpha + \sin \alpha \,)$$

$$mx'' - qx = - qb + mg \, (\mu \cos \alpha + \sin \alpha \,) \tag{8}$$

Kedua sisi kemudian dibagi *m* untuk menyederhanakan formula.

$$x'' - x \, (q/m) = - b \, (q/m) + g \, (\mu \cos \alpha + \sin \alpha \,) \tag{9}$$

Karena persamaan ini merupakan persamaan differensial non-homogen orde 2 (terdapat x" dan x dalam satu persamaan), maka dapat digunakan solusi berikut.[5]

$$x(t) = C_1 \cos \sqrt{q/m} \, . \, t + C_2 \sin \sqrt{q/m} \, . \, t + mg/q \, (\mu \cos \alpha + \sin \alpha \,) + b \tag{10}$$

Didapatkan formula untuk *x(t)*, namun masih ada faktor konstanta

yang perlu dipecahkan. Oleh karenya, diperlukan penyelesaian partikulir untuk mendapatkan nilai C_1 dan C_2. Caranya adalah dengan memasukkan nilai awal ke persamaan yang telah ada.

Pada titik awal, *initial value* adalah $t = 0$, $x = x_0$, dan $v = 0$, dimana v adalah turunan pertama dari x atau $v = x'$

Dengan memasukkan *initial value* pada persamaan (10) maka didapatkan :

$$x(0) = C_1 \cos 0 + C_2 \sin 0 + mg/q\ (\mu \cos \alpha + \sin \alpha) + b \qquad (11)$$

$$x = x_0$$

$$x_0 = C_1 + mg/q\ (\mu \cos \alpha + \sin \alpha) + b$$

$$C_1 = x_0 - [\ mg/q\ (\mu \cos \alpha + \sin \alpha) + b] \qquad (12)$$

Didapatkan nilai C_1 , lalu untuk mencari C_2 dapat dilakukan dengan melakukan turunan sekali terhadap $x(t)$ sehingga bisa menggunakan persamaan $v = 0$.

$$x'(t) = -[\ C_1 \sqrt{q/m}\ \sin\sqrt{q/m}\ .\ t\ ] + \qquad (13)$$
$$[\ C_2 \sqrt{q/m}\ \cos\sqrt{q/m}\ .\ t\ ] + 0$$

$$x'(0) = 0$$

$$x'(0) = -[\ C_1 \sqrt{q/m}\ \sin 0\ ] + [\ C_2 \sqrt{q/m}\ \cos 0\ \ ] + 0$$

$$x'(0) = C_2 \sqrt{q/m}$$

$$x'_0 = 0 \quad \rightarrow \quad C_2 = 0 \qquad (14)$$

Dengan didapatkannya C_1 dan C_2, maka langkah selanjutnya memasukkan konstanta persamaan (12) dan (14) ke persamaan (10).

$$x(t) = [\ x_0 - mg/q\ (\mu \cos \alpha + \sin \alpha) - b\ ] \qquad (15)$$
$$\cos \sqrt{q/m}\ .\ t + mg/q\ (\mu \cos \alpha + \sin \alpha) + b$$

Persamaan di atas adalah persamaan posisi. Untuk mendapatkan persamaan kecepatan, maka cukup diturunkan sekali.

$$x'(t) = -[\, x_0 - mg/q\,(\mu \cos\alpha + \sin\alpha) - b\,] \cdot \sqrt{q/m}\ \sin\sqrt{q/m}\ \cdot t \qquad (16)$$

Dengan demikian, maka didapatkanlah formula kecepatan saat waktu *t*, yaitu saat UAV akan lepas landas. Waktu *t* sendiri juga bisa didapatkan formulanya dengan menggunakan persamaan (15).

$$x(t) = [\, x_0 - mg/q\,(\mu \cos\alpha + \sin\alpha) - b] \quad \cos\sqrt{q/m}\ \cdot t + mg/q\,(\mu \cos\alpha + \sin\alpha) + b$$

$$x(t) - mg/q\,(\mu \cos\alpha + \sin\alpha) - b = [\, x_0 - mg/q\,(\mu \cos\alpha + \sin\alpha) - b\ ] \cos\sqrt{q/m}\ \cdot t$$

$$\cos\sqrt{q/m}\ \cdot t = \frac{x(t) - mg/q\,(\mu \cos\alpha + \sin\alpha) - b}{x0 - mg/q\,(\mu \cos\alpha + \sin\alpha) - b}$$

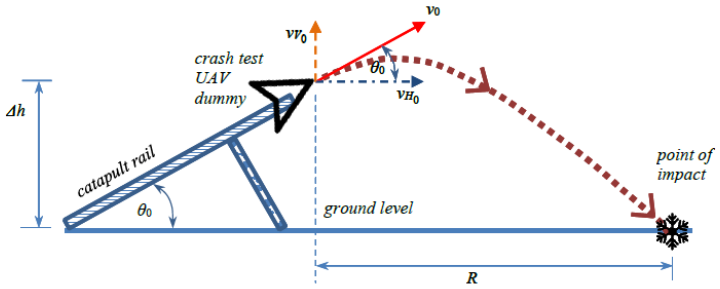
$$t = \sqrt{m/q}\ \arccos\frac{x(t) - mg/q\,(\mu \cos\alpha + \sin\alpha) - b}{x0 - mg/q\,(\mu \cos\alpha + \sin\alpha) - b} \qquad (17)$$

Persamaan *v(t)* dan *t* akan digunakan untuk formula sebelum peluncuran, dengan tujuan untuk mendapatkan *initial velocity* saat pesawat lepas landas dari peluncur.

3.3. Kalkulasi Prediksi Lintasan

Setelah mendapatkan formula kecepatan saat peluncuran, maka langkah selanjutnya adalah menentukan formula dari prediksi lintasan itu sendiri, sekaligus juga menentukan formula untuk mendapatkan percepatan. Formula prediksi lintasan akan banyak menggunakan parameter dari UAV yang digunakan, berbanding terbalik dengan saat peluncuran yang menggunakan parameter dari konfigurasi peluncur UAV.

Asumsi awal yang dilakukan disini adalah bahwa pergerakan UAV (dengan asumsi bahwa UAV tidak memiliki motor / hanya *dummy* UAV) menyerupai konsep gerak parabola.



Gambar 3.5. Gambaran kasar skema pergerakan UAV[18]

Merujuk dari rumus yang ada, maka dapat diketahui nilai *x* dan *y* dari setiap satuan waktu.

$$x = x_0 + v_0 \cos \theta \, t + at$$

$$y = y_0 + v_0 \sin \theta \, t - \tfrac{1}{2} gt^2$$

Perbedaannya dengan konsep gerak parabola, disini kecepatan pada sumbu *x* akan tetap memiliki percepatan karena apabila tidak ada faktor percepatan, maka tidak akan ada perubahan dari kecepatan awal.

Percepatan dapat didapatkan dengan menggunakan rumus :

$$F = m \, . \, a$$

dimana *F* merupakan gaya dorong pesawat atau dalam aerodinamika dikenal dengan istilah *thrust*, *m* adalah massa pesawat, dan *a* adalah percepatan. Untuk mendapatkan percepatan, maka cukup merubah rumusnya.

$$a = F / m$$

Perhitungan untuk *thrust* sendiri diperlukan untuk mendapatkan parameter kedua sehingga percepatan dapat diketahui. Ada beberapa parameter tambahan yang diperlukan, yaitu RPM dari motor, kemudian diameter dan *pitch* dari propeller serta daya atau *power* dari motor setelah dikalikan dengan efisiensi.[22]

Untuk memudahkan mendapatkan persamaan *thrust* dari fungsi propeler dan motor, maka digunakan persamaan yang sudah ada[23]

$$F = p\ x\ \frac{\pi(0.0254.d)^2}{4}\ [\ (\ RPM\ x\ 0.0254\ x\ pitch\ x\ 1min/60s\ )^2 - (\ RPM\ x\ 0.0254\ x\ pitch\ x\ 1min/60s\ )\ x\ V_0\ ]\ x\ (\ \frac{d}{3.29546\ x\ pitch}\ )^{1.5}$$

Dengan didapatkan formula *thrust*, maka percepatan pun bisa didapatkan. Maka langkah selanjutnya hanyalah menentukan kapan dan dimana titik *turning point* atau titik ketika pesawat bisa menukik naik.

Sesuai dengan konsep aerodinamika, agar pesawat dapat terbang stabil dan mengudara, maka *lift* harus sama dengan *weight*.

$$L = W$$

Rumus *lift* dan *weight* pun dimasukkan.

$$C_L\ .\ S\ .\ (\ ½\ .\ p\ .\ v^2) = m\ .\ g$$

dimana *S* adalah luas sayap pesawat, C_L adalah koefiien dari gaya *lift*, *p* adalah tekanan udara pada ketinggian tertentu (menggunakan nilai umum yaitu 1.225), dan *v* adalah kecepatan pesawat pada waktu pengambilan data, sehingga kecepatan akan terus berubah seiring dengan adanya percepatan. Dengan kata lain, pesawat akan dapat terbang naik dan tidak menukik turun apabila *lift* sudah tidak lebih kecil dari *weight*, yang mana kondisi ini sangat bergantung pada perubahan kecepatan dari percepatan yang didapat sebelumnya. Posisi titik ini yang akan direferensikan sebagai *turning point*.

Prediksi lintasan setelah melewati titik *turning point* sebenarnya sangat bergantung dari kendali pilot, sehingga cukup rumit untuk dibuat prediksi lintasannya. Sehingga, prediksi lintasan yang dimaksud disini hanyalah sampai UAV mampu melewati *turning point*. Melewati *turning point* maka prediksi lintasan yang dibuat hanya akan dibuat dengan menggunakan asumsi umum tentang lintasan linear dari kenaikan ketinggian UAV setiap waktunya, yaitu dengan asumsi sudut kenaikan konstan sebesar 25º.

Setelah mendapatkan formula kalkulasi, maka selanjutnya dilakukan pendataan terhadap parameter-parameter yang dimaksud untuk mempermudah pemahaman dalam menerjemahkan ke dalam program nantinya.

- ***(T) – Tension***

Tension merupakan parameter yang ada dalam remot pengendali peluncur UAV. Pada remot, tertuliskan *tension* dari peluncur UAV yang diukur oleh *load cell* yang diletakkan tepat di bawah *cradle* UAV.

Gambar 3.6. Remot pengendali peluncur UAV

Satuan yang digunakan disini masih dalam kilogram. Untuk mendapatkan dalam satuan newton, maka dikalikan oleh gravitasi.

$$F = Tension \times 9.81$$

Hasil perkalian *tension* dengan gravitasi adalah gaya tarik yang dialami oleh *cradle*, yang nantinya dapat digunakan untuk mencari perpanjangan karet pelontar dengan membaginya dengan koefisien elastisitas.

- ***(q)* – Koefisien Elastisitas**

Penggunaan *bungee cord* pada peluncur UAV adalah tak lain karena ia bisa memanjang atau meregang dan menghasilkan gaya. Saat *cradle* UAV dikunci posisi, maka karet ini akan menarik *cradle* dan meregang sesuai dengan perintah pada remot. Saat kunci dilepas, maka gaya yang dihasilkan dari karet ini akan melontarkan UAV untuk lepas landas.

Setiap benda memiliki koefisien elastisitas, termasuk *bungee cord*. Pada beberapa *bungee cord*, terdapat nilai koefisien elastisitas pada produk saat pembelian. Selain itu, bisa juga digunakan Hukum Hooke untuk mencari tahu nilai koefisien elastisitas benda tersebut.

Dengan menggantungkan karet pada suatu bidang datar yang mengambang kemudian memberikan beban pada ujung yang lain, maka dapat didata nilai koefisien elastistas yang dimaksud.

Sesuai dengan rumus Hooke :

$$F = k \, . \, x$$

Maka :

$$k = F / x$$

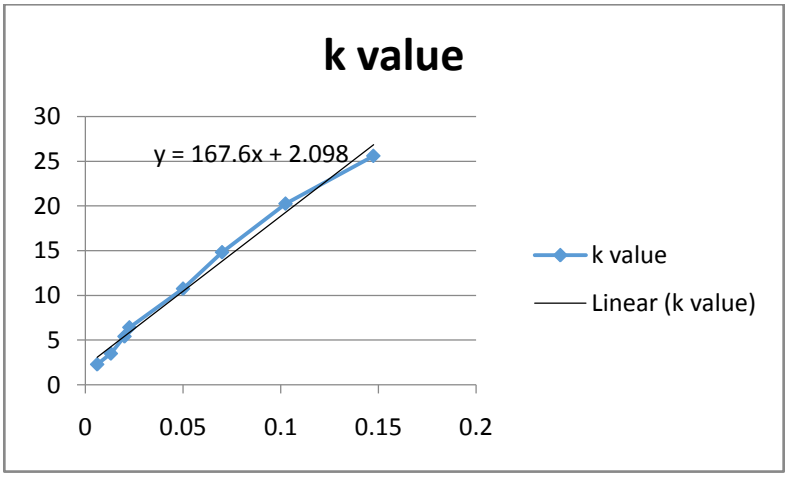
Gambar 3.7. Percobaan untuk mengetahui nilai koefisien elastisitas

Untuk mendapatkan nilai k, dilakukan percobaan dengan berbagai jenis massa dan dilakukan pendataan terhadap perpanjangan dari tiap beban yang diberikan.

Setelah didapatkan data, kemudian dilakukan *plotting* dan digunakan regresi linier untuk mendapatkan nilai *k* yang sebenarnya.

Tabel 3.1. Perpanjangan karet terhadap massa yang diberikan

Massa *(kg)*	Gaya Berat *(N)*	Perpanjangan *(m)*	*k*
0.235	2.30535	0.006	384.225
0.358	3.51198	0.013	270.152
0.553	5.42493	0.02	271.246
0.656	6.43536	0.0225	286.016
1.097	10.76157	0.05	215.231
1.512	14.83272	0.07	211.896
2.065	20.25765	0.1025	197.635
2.609	25.59429	0.1475	173.520



Gambar 3.8. Grafik nilai koefisien elastisitas

Maka didapatkan nilai *k* 167.6x dengan menganulir nilai konstanta 2.098 yang terbilang kecil (~2%).

- ***(m)* – Massa**

Massa merupakan parameter yang berubah-ubah, tergantung jenis UAV yang dinaikkan. Massa ini sendiri dibagi menjadi 2, yaitu massa UAV dan massa *cradle*.

Massa *cradle* adalah massa yang tidak berubah karena hanya satu jenis. *Cradle* pada peluncur yang digunakan memiliki massa 3.919 kg. Sedangkan untuk massa UAV akan bervariasi tergantung dari jenis UAV. Saat peluncuran, UAV akan diletakkan pada *cradle* sehingga massa yang dimaksud disini adalah penjumlahan massa *cradle* dan UAV.

$$m = m_{UAV} + m_{CRD}$$

- ***(x_0) – Elongated Cord Length***

Perpanjangan karet adalah nilai yang berubah-ubah juga, tergantung dari berapa *tension* yang digunakan pada peluncur UAV. Rumus Hooke sebelumnya yang sudah didapatkan akan digunakan untuk mencari tahu perpanjangan karet dari beban yang diberikan.

$$F = k \, . \, x$$

$$x = F / k$$

Rumus Hooke di atas hanyalah mencari perpanjangan. Oleh karenanya, untuk mendapatkan panjang karet yang meregang, ditambahkan dengan panjang awal dari karet tersebut :

$$x_0 = b + x$$

b ini sendiri adalah panjang awal karet atau panjang karet sebelum meregang.

- ***(α)* – Sudut Peluncuran**

Sama halnya dengan *tension*, sudut peluncuran tertera pada remot pengendali.

Gambar 3.9. Remot pengendali peluncur UAV

Peluncur UAV yang digunakan disini memang awalnya dirancang untuk memiliki sudut sekitar 10 - 15º[4]. Namun apabila pengguna membutuhkan sudut lebih, maka dapat diakali dengan menambah atau meninggikan kaki depan peluncur UAV sehingga sudut peluncuran akan lebih besar

.

- **Luas Permukaan UAV**

Luas permukaan UAV digunakan untuk mengetahui nilai *lift* dari UAV saat mengudara. Nilai ini juga akan berubah-ubah tergantung jenis UAV yang digunakan. Parameter satu ini akan sulit untuk didapatkan dikarenakan bentuk UAV yang bermacam-macam sehingga tidak mungkin untuk menerjemahkan bentuk UAV ke dalam perhitungan matematis luas benda (persegi, segitiga, dan lain lain). Untuk itu, nantinya akan dibuat perangkat keras pendukung yang dapat membaca luas permukaan UAV.

- ***(C_L) – Lift Coefficient***

Lift Coefficient merupakan konstanta yang digunakan untuk menghitung *lift* dari pesawat. *Lift coefficient* ini biasanya didapat dan dituliskan dalam buku manual panduan pesawat dari produsen pesawat dalam bentuk grafik. *Lift coefficient* ini dipengaruhi oleh berbagai macam faktor, mulai dari bentuk pesawat, luas sayap pesawat, jenis

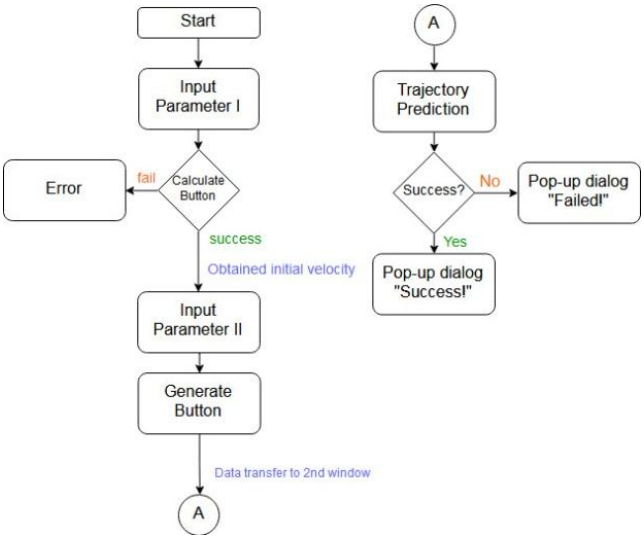
airfoil, hingga *angle of attack* dari ketinggian pesawat.

Untuk mempermudah perhitungan, maka *lift coefficient* akan ditetapkan ke nilai *default* yaitu 0.5 dengan asumsi bahwa *angle of attack* adalah 5º[6] dan C_L sebesar ~0.11 setiap derajatnya[7]. Selain itu, sebenarnya faktor aerodinamika bisa dianulirkan pada peluncuran dengan kecepatan di bawah 30m/s sehingga faktor C_L ini tidak akan banyak berpengaruh, hanya saja tetap diperlukan untuk mendapatkan nilai *lift* sebagai perbandingan terhadap *weight* dalam waktu tertentu.

3.4. Perangkat Prediksi Lintasan

Dengan didapatkannya formula kalkulasi dari kedua kondisi yaitu formula untuk peluncuran dan formula prediksi lintasan, maka selanjutnya adalah merancang dan membuat perangkat yang digunakan untuk memprediksi lintasan. Perangkat akan dibagi menjadi 2 macam, yaitu perangkat berbasis aplikasi yang dapat dijalankan di laptop serta perangkat berbasis Arduino.

3.4.1. Aplikasi Prediksi Lintasan



Gambar 3.10. *Flowchart* program.

Perangkat prediksi lintasan yang pertama adalah aplikasi yang dibuat menggunakan QT Creator. Dalam pembuatan aplikasi ini, rencananya akan dibuat dua halaman. Halaman pertama berisi tentang form-form data yang diperlukan dan melakukan kalkulasi untuk formula kalkulasi saat peluncuran. Hasil kalkulasi ini nantinya dapat diakses oleh halaman kedua untuk perhitungan formula kalkulasi prediksi lintasan. Halaman kedua akan lebih berfokus untuk menampilkan hasil prediksi lintasan dalam bentuk grafis berdasarkan data dari hasil kalkulasi prediksi lintasan. Juga, halaman kedua ini hanya akan muncul apabila pada halaman pertama telah ditekan tombol *'Generate'*.

3.4.1.1. *Pembuatan Halaman 1*

Halaman pertama akan menampilkan tatap muka utama dari perangkat lunak yang dirancang. Hal yang harus ada adalah tempat mengisi data untuk data-data yang diperlukan. Selain itu, perlu dibuat juga sebuah koneksi antara halaman pertama dengan kedua sehingga halaman kedua dapat mengakses data-data dari halaman pertama. Untuk kasus ini, parameter-parameter terkait dapat didefinisikan sebagai *public* pada *header*.

- ***Header***

Dalam QT Creator, terdapat beberapa pilihan *Widget* yang dapat digunakan untuk memasukkan data, diantaranya adalah QLineEdit dan QTextEdit. Pada umumnya, teks dengan jumlah karakter banyak akan menggunakan QTextEdit, dan QLineEdit lebih sering dimanfaatkan untuk data masukan yang tidak banyak (semisal angka berupa usia, tinggi badan, dan lain lain). Untuk itu, disini yang akan digunakan adalah QLineEdit.

Merujuk kepada rumus yang ada sebelumnya, ada sejumlah parameter yang perlu dimasukkan. Sebelum memulai membuat program, maka terlebih dahulu didefinisikan QLineEdit dan semua parameter lain yang akan digunakan pada header mainwindow (mainwindow.h).

```
public:
    explicit MainWindow(QWidget *parent = nullptr);
   ~MainWindow();
    void setupUi();
    void calculate();
    void calculate2();
    double v_output;
    double acc_output;
```

```
    QLineEdit *line3; // angle
     QLineEdit *line4; // mass
     QLineEdit *line5; // velocity
     QLineEdit *line_b; // wingspan
     QLineEdit *line_e; // acc2

public slots:
     void on_showButton_clicked(){
         this->calculate2();
     }
     void on_calculateButton_clicked(){
         this->calculate();
     }
```

Sesuai dengan apa yang dipaparkan sebelumnya, beberapa parameter seperti sudut peluncuran, massa, kecepatan, dan data-data yang terkait dengan kalkulasi prediksi lintasan didefinisikan sebagai *public data* agar dapat diakses oleh halaman kedua.

Selain itu, ada juga yang dinamakan *slots*. *Slots* pada QT Creator ini berkaitan erat dengan *signals*¸dimana kedua hal ini adalah sejenis protokol komunikasi yang memungkinkan antar halaman untuk dapat bertukar data atau melakukan sesuatu atas dasar perintah sebelumnya. Mudahnya, *signals* adalah kegiatan yang akan memicu *slots* untuk bekerja (dalam bahasa C sederhana : *if signals -> then do slots)*. *Signals* dan *slots* akan dihubungkan dengan perintah *connect* yang akan dibahas pada program utama.

Sisa dari parameter yang hanya diperlukan untuk formula kalkulasi saat peluncuran dapat didefinisikan dalam *private data*. Selain QLineEdit, diperlukan juga beberapa *widget* lain seperti QPushButton (untuk menampilkan halaman kedua) dan QFrame *vertical line* untuk membantu nanti dalam menempatkan *widget-widget* terkait. Juga diperlukan QLabel untuk menandai atau menamai setiap QLineEdit sesuai bagiannya. Untuk itu, satu QLineEdit akan berpasangan dengan satu QLabel.

```
private:
     QPushButton *showButton;
     QPushButton *calculateButton;
     SecondWindow *secondWindow;
     QLabel *label1;
     QLabel *label2;
     QLabel *label2a;
     QLabel *label3;
     QLabel *label4;
```

```
    QLabel *label5;
    QLabel *label6;
    QLabel *label7;
    QLabel *label8;
    QLabel *label8a;
    QLabel *label9;
    QLabel *label10;
    QLabel *label11;
    QLabel *label12;
    QLabel *label13;
    QLineEdit *line1;
    QLineEdit *line2;
    QLineEdit *line2a;
    QLineEdit *line6;
    QLineEdit *line7;
    QLineEdit *line8;
    QLineEdit *line8a;
    QLineEdit *line9;
    QLineEdit *line10;
    QLineEdit *line11;
    QLineEdit *line12;
    QLineEdit *line13;
    QFrame *verticalLine;
    QLabel *label_a;
    QLabel *label_a1;
    QLabel *label_b;
    QLabel *label_c;
    QLabel *label_d;
    QLabel *label_e;
    QLabel *label_f;
    QLineEdit *line_a;
    QLineEdit *line_a1;
    QLineEdit *line_c;
    QLineEdit *line_d;
    QLineEdit *line_f;
};
```

- ***Main Program - Interface***

Dalam program utama (mainwindow.cpp), *widget* yang telah didefinisikan pada *header* kemudian dibuat dengan perintah *new.* Seperti yang telah dipaparkan sebelumnya, QLineEdit akan berpasangan dengan QLabel untuk menamai QLineEdit terkait.

Pengguna dapat mengisi data pada QLineEdit terkait yang merupakan variabel bebas, yaitu *tension*, *angle*, *UAV mass*, dan *cords usage*. Untuk parameter lain yang bersifat tetap, maka QLineEdit terkait dapat diatur menjadi *read-only*, agar tidak dapat dirubah oleh pengguna namun tetap ditampilkan untuk menunjukkan parameter yang

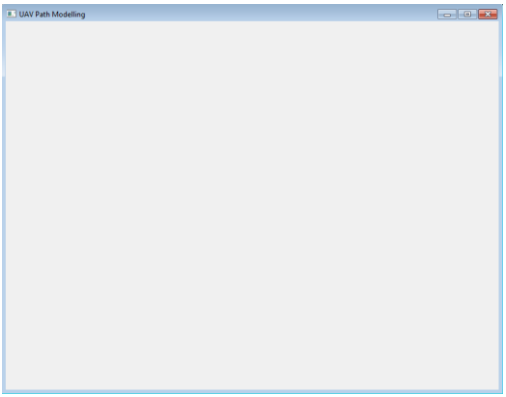
bersangkutan. Juga, untuk tipe QLineEdit yang bersifat *read-only*, tampilannya akan dirubah menjadi warna abu-abu sehingga antara QLineEdit yang dapat diisi dengan yang tidak dapat dibedakan oleh pengguna. Mewarnai QLineEdit bisa dilakukan dengan menggunakan QPalette.

```
QPalette *palette = new QPalette();
palette->setColor(QPalette::Base,Qt::lightGray);
palette->setColor(QPalette::Text,Qt::white);

QPalette *palette2 = new QPalette();
palette2->setColor(QPalette::Base,Qt::blue);
palette2->setColor(QPalette::Text,Qt::white);

label1 = new QLabel();
label1->setText("Cords Usage");
line1 = new QLineEdit();

label2 = new QLabel();
label2->setText("Tension on Controller (kg)");
line2 = new QLineEdit();

label2a = new QLabel();
label2a->setText("Propulsive Force (N)");
line2a = new QLineEdit();
line2a->setReadOnly(true);
line2a->setPalette(*palette);

label3 = new QLabel();
label3->setText("Angle");
line3 = new QLineEdit();

label4 = new QLabel();
label4->setText("UAV Mass (kg)");
line4 = new QLineEdit();

label6 = new QLabel();
label6->setText("Coefficient of Elasticity");
line6 = new QLineEdit();
line6->setText("169.44");
line6->setReadOnly(true);
line6->setPalette(*palette);

label7 = new QLabel();
label7->setText("Initial Cord Length (m)");
line7 = new QLineEdit();
line7->setText("0.7");
line7->setReadOnly(true);
```

```
line7->setPalette(*palette);

label8 = new QLabel();
label8->setText("Cord Elongation (m)");
line8 = new QLineEdit();
line8->setReadOnly(true);
line8->setPalette(*palette);

label8a = new QLabel();
label8a->setText("Elongated Cord Length (m)");
line8a = new QLineEdit();
line8a->setReadOnly(true);
line8a->setPalette(*palette);

label9 = new QLabel();
label9->setText("Cradle Mass (kg)");
line9 = new QLineEdit();
line9->setText("3.919");
line9->setReadOnly(true);
line9->setPalette(*palette);

label10 = new QLabel();
label10->setText("Total Mass (kg)");
line10 = new QLineEdit();
line10->setReadOnly(true);
line10->setPalette(*palette);

label11 = new QLabel();
label11->setText("Coefficient of Friction");
line11 = new QLineEdit();
line11->setText("0.1");
line11->setReadOnly(true);
line11->setPalette(*palette);

label12 = new QLabel();
label12->setText("Gravitational Force");
line12 = new QLineEdit();
line12->setText("9.81");
line12->setReadOnly(true);
line12->setPalette(*palette);

label13 = new QLabel();
label13->setText("Launching Time (s)");
line13 = new QLineEdit();
line13->setReadOnly(true);
line13->setPalette(*palette2);

label5 = new QLabel();
label5->setText("Result of Velocity Calculation (m/s)");
line5 = new QLineEdit();
```

```
line5->setReadOnly(true);
line5->setPalette(*palette2);

calculateButton = new QPushButton("Calculate");

verticalLine = new QFrame();
verticalLine->setFrameShape(QFrame::VLine);
verticalLine->setFrameShadow(QFrame::Sunken);

label_a = new QLabel();
label_a->setText("Prop. Diameter (inch)");
line_a = new QLineEdit();

label_a1 = new QLabel();
label_a1->setText("Prop. Pitch (inch)");
line_a1 = new QLineEdit();

label_b = new QLabel();
label_b->setText("Wingspan (m2)");
line_b = new QLineEdit();

label_c = new QLabel();
label_c->setText("Motor RPM");
line_c = new QLineEdit();

label_d = new QLabel();
label_d->setText("Lift Coefficient");
line_d = new QLineEdit();
line_d->setText("0.6");
line_d->setReadOnly(true);
line_d->setPalette(*palette);

label_e = new QLabel();
label_e->setText("Acceleration (m/s2)");
line_e = new QLineEdit();
line_e->setReadOnly(true);
line_e->setPalette(*palette2);

label_f = new QLabel();
label_f->setText("Thrust (N)");
line_f = new QLineEdit();
line_f->setReadOnly(true);
line_f->setPalette(*palette2);

showButton = new QPushButton("Generate");
```

Program sejauh ini telah membuat *widget* yang diperlukan, namun apabila ditampilkan maka tidak ada yang muncul, hanya halaman kosong.



Gambar 3.11. Tampilan sementara antarmuka aplikasi

Ini dikarenakan *widget* harus masuk dalam *layout* terlebih dahulu supaya dapat tampil. Oleh karenanya, langkah selanjutnya adalah melakukan *layout* untuk semua parameter, dengan parameter untuk kalkulasi saat peluncuran dan prediksi lintasan akan dipisah untuk memudahkan pengguna memahami.

Terdapat dua jenis *layout*, yaitu vertikal dan horizontal. Parameter terkait yang perlu disejajarkan secara horizontal harus di-*layout* secara horizontal terlebih dahulu sebelum nantinya masuk dalam *layout* vertikal. Contohnya adalah *propulsive force* dan *tension* yang saling berkaitan.

```
QVBoxLayout *vboxa = new QVBoxLayout();
vboxa->addWidget(label2);
vboxa->addWidget(line2);

QVBoxLayout *vboxb = new QVBoxLayout();
vboxb->addWidget(label2a);
vboxb->addWidget(line2a);

QVBoxLayout *vboxc = new QVBoxLayout();
vboxc->addWidget(label7);
vboxc->addWidget(line7);

QVBoxLayout *vboxd = new QVBoxLayout();
vboxd->addWidget(label8);
vboxd->addWidget(line8);
```

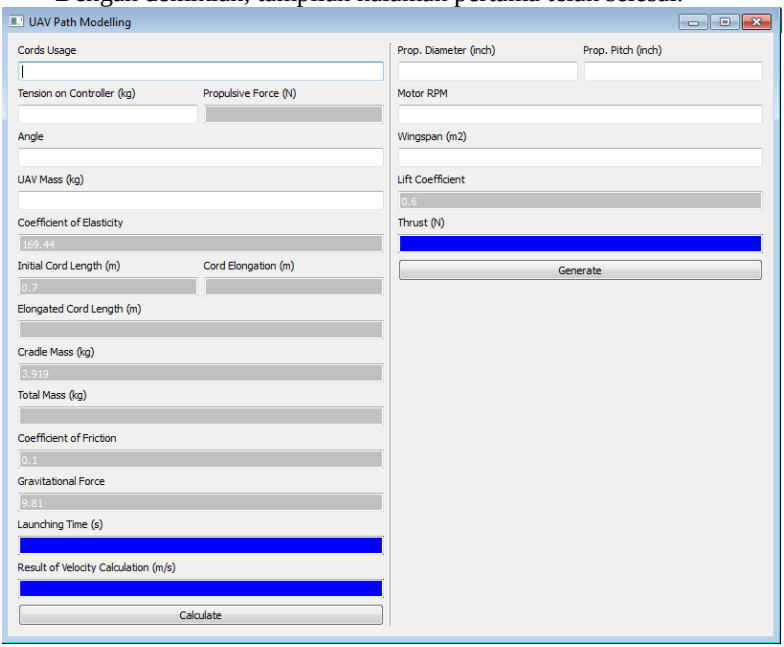
```
QVBoxLayout *vboxe = new QVBoxLayout();
vboxe->addWidget(label_a);
vboxe->addWidget(line_a);

QVBoxLayout *vboxf = new QVBoxLayout();
vboxf->addWidget(label_a1);
vboxf->addWidget(line_a1);

QHBoxLayout *hbox0 = new QHBoxLayout();
hbox0->addLayout(vboxa);
hbox0->addLayout(vboxb);

QHBoxLayout *hbox1 = new QHBoxLayout();
hbox1->addLayout(vboxc);
hbox1->addLayout(vboxd);

QHBoxLayout *hbox2 = new QHBoxLayout();
hbox2->addLayout(vboxe);
hbox2->addLayout(vboxf);

QVBoxLayout *vbox1 = new QVBoxLayout();
vbox1->addWidget(label1);
vbox1->addWidget(line1);
vbox1->addLayout(hbox0);
vbox1->addWidget(label3);
vbox1->addWidget(line3);
vbox1->addWidget(label4);
vbox1->addWidget(line4);
vbox1->addWidget(label6);
vbox1->addWidget(line6);
vbox1->addLayout(hbox1);
vbox1->addWidget(label8a);
vbox1->addWidget(line8a);
vbox1->addWidget(label9);
vbox1->addWidget(line9);
vbox1->addWidget(label10);
vbox1->addWidget(line10);
vbox1->addWidget(label11);
vbox1->addWidget(line11);
vbox1->addWidget(label12);
vbox1->addWidget(line12);
vbox1->addWidget(label13);
vbox1->addWidget(line13);
vbox1->addWidget(label5);
vbox1->addWidget(line5);
vbox1->addWidget(calculateButton);
vbox1->addStretch();

QVBoxLayout *vbox2 = new QVBoxLayout();
vbox2->addLayout(hbox2);
```

```
vbox2->addWidget(label_c);
vbox2->addWidget(line_c);
vbox2->addWidget(label_b);
vbox2->addWidget(line_b);
vbox2->addWidget(label_d);
vbox2->addWidget(line_d);
vbox2->addWidget(label_f);
vbox2->addWidget(line_f);
vbox2->addWidget(showButton);
vbox2->addStretch();
```

Setelah semuanya masuk dalam *layout*, maka langkah terakhir adalah melakukan *set* pada halaman pertama, yaitu *this*.

```
this->setLayout(layout);
```

Dengan demikian, tampilan halaman pertama telah selesai.



Gambar 3.12. Tampilan akhir antarmuka aplikasi halaman 1

- ***Main Program – Calculation***

Formula kalkulasi yang telah didapat sebelumnya kemudian diterjemahkan ke dalam bentuk program. Proses kalkulasi masuk dalam program pada mainwindow yaitu *MainWindow::Calculate()* dan *MainWindow::Calculate2()*.. Nantinya, program *Calculate* ini hanya akan berjalan apabila tombol *'Calculate*' ditekan, tentu saja dengan menggunakan proses *connect* terlebih dahulu, sedangkan *Calculate2* akan berjalan setelah tombol *'Generate'* ditekan. Seperti yang telah dipaparkan sebelumnya, ini adalah perhitungan yang terpisah yaitu saat peluncuran dan prediksi lintasan. Dengan demikian, program ini dapat dijalankan berkali-kali tanpa harus membuka-tutup program.

Pada fungsi *Calculate*, beberapa parameter yang sudah merupakan variabel tetap akan didefinisikan dengan nilai tetapnya, sedangkan yang merupakan variabel bebas akan mendapat nilainya dari proses mengkonversi data dari QLineEdit terkait.

```
    double cords = line1->text().toDouble(); //raw
    double tension = line2->text().toDouble(); //raw
    double angle = line3->text().toDouble(); //raw
    double mass_uav = line4->text().toDouble(); //raw
    double length_elong = line8->text().toDouble();
//calculated
    double mass_total = line10->text().toDouble();
//calculated
    double mass_crad = 3.919;
    double elasticity = 338.2;
    double length_init = 0.7;
    double length_stretch = 0.0;
    double friction = 0.1;
    double gravity = 9.81;
    double velocity = 0.0;
    double time = 0.0;
    double nr = 0.0;
    double phi = 3.1415;
    double force = 0.0;
```

Merujuk pada rumus sebelumnya, maka waktu dan kecepatan dapat diperoleh.

```
    force = tension * gravity;
    mass_total = mass_uav + mass_crad;
    length_elong = (force/cords)/ elasticity ;
    length_stretch = length_init + length_elong;
```

```
    time = qrt(mass_total/elasticity)) *
acos(((length_init)-((mass_total*gravity/elasticity)*
(friction*cos(angle*phi/180)+sin(angle*phi/180)))-length
_init)/(length_stretch-((mass_total*gravity/elasticity)*
(friction*cos(angle*phi/180)+sin(angle*phi/180)))-length
_init));

    velocity        =        (length_stretch        -
((mass_total*gravity/elasticity)*(friction*cos(angle*phi
/180)+sin(angle*phi/180)))-              length_init)*
(sqrt(elasticity/mass_total))*(sin((sqrt(elasticity/mass
_total))*time));
```

Setelah itu, nilai yang didapat dari proses kalkulasi akan dimasukkan ke QLineEdit terkait, sehingga pengguna dapat melihat hasil akhir dari kalkulasi saat peluncuran.

```
QString s0,s1,s2,s3,s4,s5;
line2a->setText(s0.setNum(force));
line8->setText(s1.setNum(length_elong));
line8a->setText(s2.setNum(length_stretch));
line10->setText(s3.setNum(mass_total));
line5->setText(s4.setNum(velocity));
line13->setText(s5.setNum(time));
```

Hampir sama dengan perhitungan saat peluncuran, perhitungan prediksi lintasan akan menghitung *thrust* dan percepatan yang kemudian nilainya juga akan ditransfer ke halaman kedua.

```
double diameter = line_a->text().toDouble();
double pitch = line_a1->text().toDouble();
double rpm = line_c->text().toDouble();
double mass_uav = line4->text().toDouble(); //raw
double force2 = (1.225 *
((3.14*(0.0254*diameter)*(0.0254*diameter))/4) *
(rpm*0.0254*pitch/60)*(rpm*0.0254*pitch/60)
*(sqrt((diameter/(3.29547*pitch))*(diameter/(3.29547*pitch))
*(diameter/(3.29547*pitch)))));
double acc2 = 0;
if(force2>0){
   acc2 = force2 / mass_uav;
}
QString s0,s1;
line_e->setText(s0.setNum(acc2));
line_f->setText(s1.setNum(force2));

acc_output = acc2;
```

- ***Main Program – Connect***

Langkah terakhir dari halaman pertama adalah memastikan bahwa halaman pertama dan halaman kedua terkoneksi dengan baik, begitu juga dengan *slots* dengan *signals* yang dirancang yang ada pada halaman pertama.

Yang pertama, yaitu *connect* untuk menghubungkan antara tombol *'Calculate'* dengan proses *MainWindow::Calculate()*. Yang kedua, yaitu *connect* agar saat tombol *'Generate'* ditekan, maka halaman kedua akan muncul.

```
connect(calculateButton,SIGNAL(clicked()),
        this,SLOT(on_calculateButton_clicked()));

connect(showButton,SIGNAL(clicked()),
        this,SLOT(on_showButton_clicked()));
```

Connect di atas merupakan *connect* pada halaman yang sama. Untuk *connect* antar halaman, maka peletakan perintah *connect* harus pada konstruktor mainwindow.

```
MainWindow::MainWindow(QWidget *parent) :
    QWidget(parent)
{
    this->setupUi();
    secondWindow = new SecondWindow();

    connect(line5,SIGNAL(textChanged(QString)),
    secondWindow,SLOT(receiveData(QString)));

    connect(line3,SIGNAL(textChanged(QString)),
    secondWindow,SLOT(receiveAngle(QString)));

    connect(line4,SIGNAL(textChanged(QString)),
    secondWindow,SLOT(receiveMass(QString)));

    connect(line_a,SIGNAL(textChanged(QString)),
    secondWindow,SLOT(receiveAcc(QString)));

    connect(line_b,SIGNAL(textChanged(QString)),
    secondWindow,SLOT(receiveWingspan(QString)));
}
```

Dengan adanya *connect* ini, maka apabila QLineEdit yang terkait diisi atau nilainya berubah, maka secara otomatis akan memicu *slot* untuk bekerja. *Slot* yang dimaksud disini merupakan slot yang ada pada di halaman kedua atau didefinisikan pada *header* secondwindow. Untuk itu, pada *header* mainwindow, perlu diberikan *include* secondwindow.h agar *connect* ini dapat bekerja.

3.4.1.2. *Pembuatan Halaman 2*

Halaman kedua akan menampilkan grafis dari prediksi lintasan UAV sesuai dengan data masukan yang diberikan. Dengan kata lain, halaman kedua bertugas untuk memproses formula kalkulasi prediksi lintasan dengan mengambil data dari halaman pertama.

- ***Header***

Sama halnya dengan halaman pertama, langkah pertama yang harus dilakukan adalah mendefinisikan *widget* yang akan digunakan pada *header*. Yang perlu diperhatikan adalah jangan lupa untuk mendefinisikan *slot* yang sebelumnya ada pada *connect* untuk menghubungkan halaman pertama dan kedua sebagai *public*. Apabila didefinisikan pada *private*, maka *connect* tidak akan bekerja.

```
public:
    SecondWindow();
    ~SecondWindow();
    void setupUi();
    int height_max = 600;
    int width_max = 800;
    int height_diag = height_max - 100;
    int width_diag = width_max - 100;
    double acc2 = 0;
    double v_out2, mass2, angle2, wingspan2, force2;
    double liftcoeff2 = 0.5;
    QLineEdit *vout;
    QLineEdit *accout;
    QLineEdit *rpm;
    QLineEdit *mass;
    QLineEdit *angle;
    QLineEdit *wingspan;

public slots:
    void receiveData(QString data){
        vout = new QLineEdit();
        vout->setText(data);
        QString s1;
```

```
    v_out2 = vout->text().toDouble();
}
void receiveMass(QString data){
    mass = new QLineEdit();
    mass->setText(data);
    QString s1;
    mass2 = mass->text().toDouble();
}
void receiveAngle(QString data){
    angle = new QLineEdit();
    angle->setText(data);
    QString s1;
    angle2 = angle->text().toDouble();
}
void receiveAcc(QString data){
    acceleration = new QLineEdit();
    acceleration->setText(data);
    QString s1;
    acc2 = acceleration->text().toDouble();
}
void receiveWingspan(QString data){
    wingspan = new QLineEdit();
    wingspan->setText(data);
    QString s1;
    wingspan2 = wingspan->text().toDouble();
```

Setiap data yang diperlukan membutuhkan satu *connect* dan satu *slot*. *Slot* yang dimaksud akan mengkonversi data yang ada pada QLineEdit menjadi angka, karena *connect* hanya bisa menghubungkan *widget* yang sama. Dengan kata lain, QLineEdit yang ada pada halaman pertama akan dihubungkan dengan halaman kedua. Bedanya, QLineEdit pada halaman kedua nantinya tidak perlu dipasang pada *layout* karena yang dibutuhkan hanya datanya.

Hal lain yang perlu diperhatikan adalah halaman kedua ini dirancang untuk *fixed-size window*, dikarenakan *mapping* nantinya akan menggunakan rumus yang tetap, sehingga apabila halaman bisa dilebarkan maka *mapping* akan menjadi kacau.

Untuk *private data*, halaman kedua hanya membutuhkan satu *slot* dan beberapa *widget* yang tidak sebanyak halaman pertama. Yang berbeda adalah adanya fungsi khusus *protected* untuk *paintEvent*.

```
private slots:
    void on_closeButton_clicked();

protected:
    void paintEvent(QPaintEvent *drw);
```

```
    private:
        QPushButton *closeButton;
        QLabel *labelx;
        QLabel *labely;
        QLabel *labelp;
        QLabel *velocity;
```

QT Creator menyediakan perintah untuk menggambar objek dengan menggunakan *paintEvent*, dengan syarat bahwa semua perintah menggambar itu ada dalam *paintEvent*. Perintah menggambar pada program selain *paintEvent* tidak akan menghasilkan apapun.

- ***Main Program – Interface***

Tatap muka merupakan hal yang sangat penting pada halaman kedua, karena tujuan dari halaman kedua ini adalah menyajikan grafis dari prediksi lintasan yang dapat dipahami oleh pengguna. Rancangan tatap muka dari halaman kedua ini kurang lebih akan menampilkan grafik dengan skala 0.5m, dengan penempatan peluncur UAV pada titik (0,0), dan akan ada *mapping* dari posisi UAV dalam setiap satuan waktu yang dibutuhkan (pada kisaran 0.1s – 0.5s).

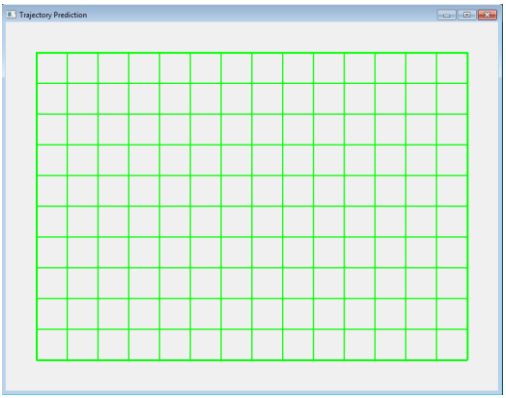
Dari kebutuhan yang telah ditentukan, maka dibuatlah program dalam *SecondWindow::paintEvent()*. Pertama-tama, yang dilakukan adalah membuat area grafik yang berjarak kurang lebih 50 pixel dari ujung halaman, atau sekitar 700x500 dari ukuran halaman yang 800x600. Area grafik yang berbentuk persegi panjang ini akan dibuat bergaris-garis dengan skala 0.5m, oleh karenanya persegi panjang ini tidak akan diwarnai bagian dalamnya atau dalam program, *brush* akan diberi jenis Qt::NoBrush. Perlu juga mendefinisikan QPen dengan tebal garis dan warna yang digunakan.

```
    QPainter painter(this);
    QPen rectpen(Qt::green);
    QBrush brush(Qt::green,Qt::NoBrush);
    rectpen.setWidth(3);
    painter.setPen(rectpen);
    QRectF
bg(QPoint(50,50),QPoint((width_max-50),(height_max-50))
;
    painter.drawRect(bg);
    painter.fillRect(bg,brush);
```

Setelah persegi panjang tergambar, maka dibuat garis skala dengan skala 0.5m atau 50 pixel, baik secara horizontal maupun vertikal.

```
double i,j;
for(i=1;i<=width_diag/50;i++){
        QPen linepen(Qt::green);
        linepen.setWidth(2);
        painter.setPen(linepen);
        j = i * 0.5;
        QPoint p1;
        p1.setX(50+(i*50));
        p1.setY((height_max-50));
        QPoint p2;
        p2.setX(p1.x());
        p2.setY(p1.y()-height_diag);
        QLine horiz(p1,p2);
        painter.drawLine(horiz);
}

for(i=1;i<=height_diag/50;i++){
        QPen linepen(Qt::green);
        linepen.setWidth(2);
        painter.setPen(linepen);
        j = i * 0.5;
        QPoint p1;
        p1.setX(50);
        p1.setY((height_max-50)-(i*50));
        QPoint p2;
        p2.setX(p1.x()+width_diag);
        p2.setY(p1.y());
        QLine vertic(p1,p2);
        painter.drawLine(vertic);
}
```

Sejauh ini, program akan menampilkan tampilan seperti gambar di bawah.



Gambar 3.13. Tampilan sementara halaman kedua

Terlihat bahwa tampilan setidaknya sudah menggambarkan sebuah grafik. Untuk memperjelas grafik, nantinya akan diberi angka-angka dengan QLabel.

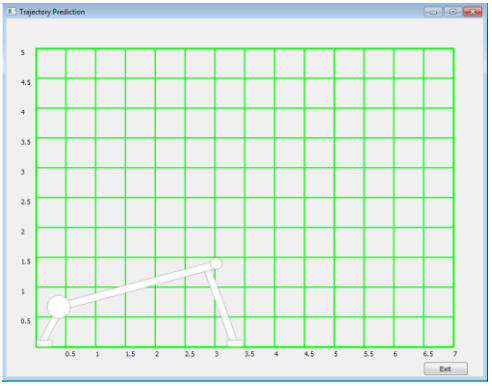
```
double i,j;
    for(i=1;i<=width_diag/50;i++){
        j = i * 0.5;
        QPoint p1;
        p1.setX(50+(i*50));
        p1.setY((height_max-45));
        QString b = QString::number(j);
        labelx = new QLabel(this);
        labelx->setText(b);
        labelx->move(p1);
    }
    for(i=1;i<=height_diag/50;i++){
        j = i * 0.5;
        QPoint p1;
        p1.setX(25);
        p1.setY((height_max-50)-(i*50));
        QString b = QString::number(j);
        labelx = new QLabel(this);
        labelx->setText(b);
        labelx->move(p1);
    }
```

Kemudian ilustrasi dari peluncur UAV diletakkan pada grafik posisi (0,0) menggunakan QPixmap.

```
QPixmap pixmap= QPixmap(":/files/uav2.png");
QPoint p2;
p2.setX(50);
p2.setY(height_max-175);
labelp = new QLabel(this);
labelp->setPixmap(QPixmap(pixmap));
labelp->move(p2);
```

Pada dasarnya, penempatan gambar pada QT Creator juga menggunakan QLabel. Hanya saja, untuk gambar, QLabel akan menggunakan QPixmap dan bukan tulisan biasa.

Sampai disini, maka tampilan dari halaman kedua bisa dilihat pada gambar berikut.



Gambar 3.14. Tampilan grafik halaman kedua

- ***Main Program – Calculation & Mapping***

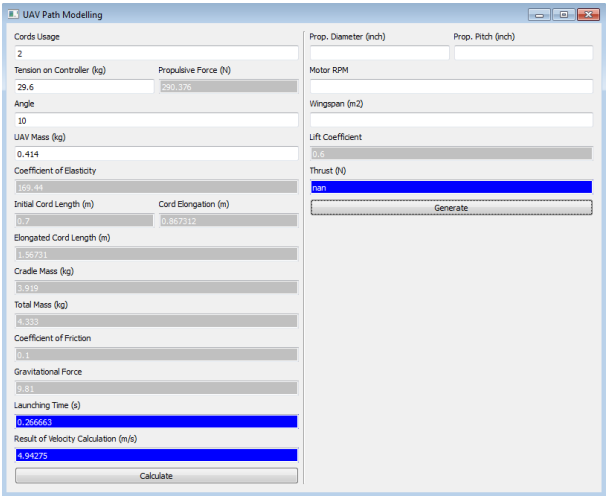
Bagian terpenting dari halaman kedua adalah bagaimana program dapat melakukan *mapping* dari posisi UAV setiap satuan waktunya, sehingga dapat menghasilkan prediksi lintasan sesuai tujuan awal. Dari setiap satuan waktunya, program akan membaca apakah kemampuan UAV untuk terbang atau *lift* sudah lebih besar dibandingkan berat atau *weight*. Apabila sampai ketinggian 0m ternyata *lift* masih lebih kecil dibandingkan *weight*, maka UAV dinyatakan jatuh atau *crash*. Namun apabila kebalikannya, maka UAV dapat dinyatakan sukses lepas landas pada ketinggian titik terendah tersebut sebelum dapat terbang naik.

Program *mapping* ini masih dalam satu bagian program *SecondWindow::paintEvent()* dikarenakan nanti di setiap posisi pada

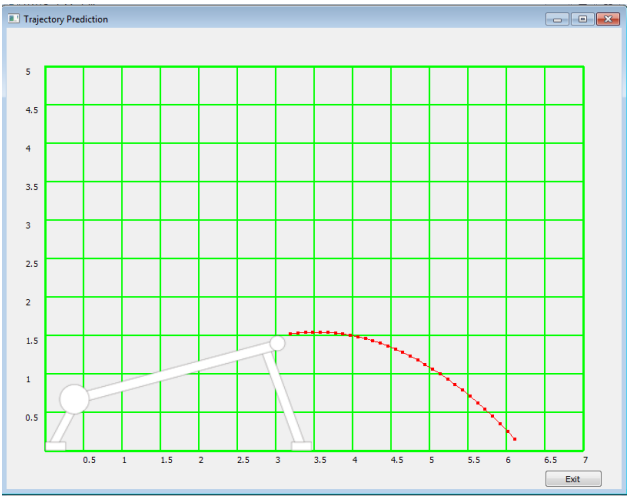
satuan waktu tertentu, akan diberikan titik sebagai penanda posisi UAV pada waktu tersebut. Titik-titik ini nantinya akan terilustrasikan sebagai prediksi lintasan yang dimaksud.

```
            double x = 0.0, vt = 0.0, xt = 0.0;
            double y = 0.0, yt = 0.0;
            double t = 0.0;
            double weight = 0.0;
            double lift = 0.0;
            double density = 1.225;
            double distance = 0.0;
            double const_ang = 25;
            double tan10 = tan(const_ang*3.1415/180);
            double time_turning = 0.0;
            int k = 0;

            for(double i=1;i<=100;i++){
                if(v_out2 >= 1){
                t = 0.02 * i;
                vt = v_out2 * cos(angle2*3.1415/180)*t + (acc2 *
t);
                x = v_out2 * cos(angle2*3.1415/180) * t;

                weight = mass2 * 9.81;
                lift = 0.5 * density * (vt * vt) * wingspan2 *
liftcoeff2;

                if(lift<weight){
                    y = ((v_out2 * sin(angle2*3.1415/180) * t ) -
(0.5 * 9.81 * t * t));
                }
                else if (lift>weight){
                    distance = xt - x;
                   y = yt - (tan10 * distance);
                   if(k==0){
                        turning_point.setX(xt);
                        turning_point.setY(yt);
                        time_turning = t-0.02;
                    }
                    k++;
                }
                if(i>1){
                   yt = y;
                   xt = x;
                   x2.setX(x1.x());
                   x2.setY(x1.y());
                }

                x1.setX(start.x()+(x*100));
                x1.setY(start.y()-(y*100));
```

```
if(x<0){
    break;
}

if(x1.x()>750){
    break;
}

if(y+1.45<0){
    break;
}

if(i>1){
    QPen ballpen(Qt::red);
    ballpen.setWidth(1);
    painter.setPen(ballpen);
    QLine trajectory1(x1,x2);
    painter.drawLine(trajectory1);
}

QPen ballpen(Qt::red);
ballpen.setWidth(4);
painter.setPen(ballpen);
painter.drawPoint(x1);
}
```



(a)



(b)

Gambar 3.15. (a).Tampilan antarmuka halaman pertama (b).Tampilan grafis prediksi lintasan halaman kedua

Dengan adanya *mapping* ini, maka satu sistem perangkat lunak yang dirancang dapat dinyatakan selesai. Gambaran dari penggunaan program secara keseluruhan dapat dilihat pada gambar berikut.

3.4.2. Perangkat Prediksi Lintasan dengan Arduino

Perangkat yang dirancang selanjutnya adalah perangkat berbasis Arduino. Perangkat ini akan memiliki pola kerja yang hampir sama dengan aplikasi sebelumnya. Hanya saja, yang ditampilkan sebagai keluaran hanya berupa indikasi apakah lintasan yang dilewati aman, beresiko ataupun berbahaya.

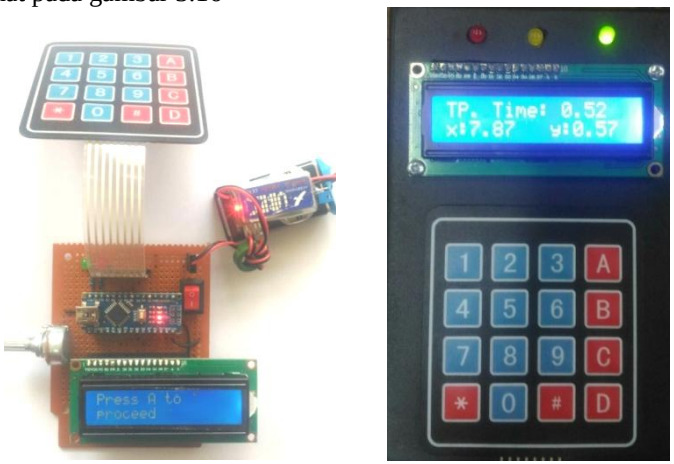
Perangkat ini akan berukuran cukup kecil dan akan memiliki LCD 16x2 sebagai alat bantu visual untuk dapat menunjukkan hasil dan proses berjalannya program. Untuk masukan, digunakan *keypad* membran 4x4 agar pengguna dapat memasukkan angka atau nilai dari parameter terkait yang dibutuhkan. Sebagai otak utama dari perangkat ini, digunakan Arduino Nano yang akan memproses masukan dan keluaran. Untuk daya, digunakan baterai 9V yang diproses oleh converter UBEC sehingga menghasilkan tegangan 5V yang menjadi masukan untuk Arduino.

Perangkat ini akan menerima data satu per satu. Setelah semua parameter dimasukkan, maka Arduino akan melakukan kalkulasi dan akan menampilkan waktu, jarak serta ketinggian dari *turning point* sekaligus akan ada lampu LED yang menunjukkan kondisi dari penerbangan dengan parameter yang diberikan.

Tabel 3.2. Kondisi yang diberikan sebagai keluaran perangkat Arduino

Kondisi	LED			Status	Penjelasan
	Merah	Jingga	Hijau		
1	OFF	OFF	ON	AMAN	Berhasil terbang, y>0.5m
2	OFF	ON	OFF	BERESIKO	Berhasil terbang, y<0.5m
3	ON	OFF	OFF	BAHAYA	Gagal terbang/jatuh

Perangkat yang telah dirancang kemudian dibuat dan hasilnya dapat dilihat pada gambar 3.16



(a) (b)

Gambar 3.16. (a) Purwarupa perangkat prediksi lintasan. (b) Perangkat prediksi lintasan dengan *casing*

Sama halnya dengan aplikasi di laptop, perangkat ini memiliki cara kerja yang kurang lebih mirip; memasukkan parameter secara berurutan mulai dari *tension*, sudut, jumlah karet, massa UAV yang kemudian menghasilkan kecepatan, kemudian memasukkan parameter UAV seperti diameter dan *pitch* propeler, RPM motor serta luas sayap sehingga dihasilkan keluaran berupa indikasi LED serta informasi *turning point* dari waktu, posisi *x* dan posisi *y*.

Menggunakan *library* dari *keypad* dan LCD, maka program dapat dengan mudah dibuat. Agar tidak terjadi *looping* saat menunggu masukan data, maka digunakan fungsi *while* sampai tombol A dipencet agar dapat berpindah ke masukan parameter selanjutnya. Modifikasi tombol pada *keypad* pun dilakukan untuk mengoptimalkan kinerja perangkat ; tombol 'D' merupakan koma, tombol '#' merupakan tombol untuk menampilkan angka yang sudah dimasukkan – setelah angka benar, lalu ditekan tombol 'A' sebagai tombol *enter*.

Gambar 3.17. Keluaran perangkat prediksi lintasan dengan Arduino

Keluaran dari perangkat yang dibuat dapat dilihat pada gambar 3.17. Terlihat bahwa ada indikator lampu LED bahwa konfigurasi ini aman untuk dijalankan dengan ketinggian *turning point* sebesar 0.71m.

3.5. *Measurement box*

Pada formula kalkulasi prediksi lintasan, terdapat beberapa parameter yang bisa dibilang cukup rumit untuk bisa diambil datanya. Salah satu parameter yang dimaksud ini adalah lebar sayap atau lebar area pesawat (*wingspan*). Untuk dapat menunjang performa dari perangkat lunak yang dirancang, maka dilakukan pula pembuatan perangkat keras yang dapat membantu untuk membaca luas area pesawat.

Perangkat keras ini nantinya akan berupa sejenis kotak besar, dimana pesawat yang ingin diketahui lebar sayapnya akan dimasukkan kedalamnya. Pada bagian atap atau atas dari kotak tersebut, dipasang kamera yang akan mengambil gambar dari pesawat. Gambar yang diambil kemudian diolah menggunakan program OpenCV untuk dapat mengetahui luasan permukaan dari area yang dimaksud.

3.5.1. Spesifikasi Alat

Measurement box dibuat menggunakan multipleks dengan ukuran awal 150x75x120 cm. Hanya saja, tinggi dari kamera ternyata belum cukup untuk dapat menangkap area luasan pesawat dengan baik, sehingga dilakukan improvisasi dengan meninggikan posisi kamera pada tinggi 145cm. Dengan kata lain, pesawat atau UAV yang dapat diukur hanyalah UAV yang memiliki lebar kurang dari 1.4m. Selain itu, juga ada lampu yang diberikan di bagian depan sebagai penerangan tambahan agar faktor cahaya bisa direduksi. Lampu ini menggunakan tegangan 220V.

(a) (b)

Gambar 3.18. (a). *Measurement box* tampak depan (b). *Measurement box* tampak samping

Kamera diletakkan pada titik tertinggi dan menghadap ke bawah, dan pemasangan kamera dilakukan sedemikian rupa sehingga bisa diatur dan digeser untuk menghasilkan tangkapan gambar terbaik.

Gambar 3.19. Kamera pada *measurement box*

Pada area tangkapan kamera, diberikan warna homogen (hijau pada gambar) untuk mempermudah dari program membedakan mana yang pesawat dan mana yang bukan.

Gambar 3.20. Area tangkapan kamera yang mayoritas berwarna hijau

Dengan menggunakan metode *threshold* pada *HSV color space*, maka kamera bisa membedakan dengan akurat hijau yang digunakan pada latar belakang dengan hijau yang ada pada UAV, kecuali jika memang kebetulan warna hijau dari keduanya benar-benar persis.

3.5.2. Pembuatan Program

Dengan menggunakan MS Visual Studio 2015 dan OpenCV, dirancanglah program untuk dapat membaca luasan area pesawat dengan menggunakan kamera.

Fitur utama yang akan digunakan dalam program ini adalah *contour*. Dalam OpenCV, *contour* atau *blob* merupakan pixel-pixel yang tergabung menjadi satu membentuk satu bentuk. Mudahnya, untuk dapat mengukur lebar area pesawat akan sejalan dengan mengukur ukuran pixel dari pesawat itu sendiri, meskipun memang bukan hanya sayap saja yang nantinya akan masuk dalam kamera. Jadi, metode pengukuran dari luas sayap ini menggunakan fungsi *findContours.*

Pada fungsi main, awalnya didefinisikan terlebih dahulu beberapa parameter yang akan digunakan, yaitu *Mat*, *element*, dan *vector*; *Mat* atau matrix yang akan digunakan untuk menyetor gambar hasil tangkapan kamera, *element* sebagai parameter untuk nanti melakukan proses pengolahan gambar, serta *vector* baik *Point* ataupun *Vec4i* karena keduanya adalah parameter dalam fungsi *findContours*.

```
        namedWindow("Ori", CV_WINDOW_NORMAL);
        resizeWindow("Ori", Size(640, 480));
        Mat frame, frameHSV, frameTHR;
        Mat element2 = getStructuringElement(MORPH_RECT,
    Size(5, 5), Point(-1, -1));
        vector<vector<Point>> contours;
        vector<Vec4i>hierarchy;
        VideoCapture cap(1);
```

Setelahnya, masuk ke dalam fungsi *loop* dimana program akan mengambil gambar dari tiap waktunya untuk pembanding. Setelah mendapat cukup data, maka nilai dari momen gambar akan dirata-rata untuk nantinya menjadi nilai akhir dari luasan pesawat yang masuk ke dalam *measurement box*. Hal ini dilakukan karena mungkin saja dalam satu tangkapan gambar, terjadi pergeseran atau ada pantulan dari cahaya yang menyebabkan luas area pesawat tidak tertangkap sepenuhnya. Dengan demikian, *noise* dapat dicegah.

```
        cap >> frame;
        cvtColor(frame, frameHSV, CV_BGR2HSV);
        inRange(frameHSV, Scalar(40, 30, 30), Scalar(80, 255,
255), frameTHR);
        dilate(frameTHR, frameTHR, element2);
```

Untuk membedakan pesawat dengan latar belakang, digunakan fungsi *inRange* pada daerah warna hijau muda setelah sebelumnya gambar dikonversi ke dalam dimensi warna HSV. Penentuan daerah *inRange* dilakukan secara hati-hati, karena semakin luas daerah *inRange* maka bisa jadi warna hijau lain akan terbaca. Namun semakin sempit daerah *inRange*, faktor cahaya akan banyak mempengaruhi. Meskipun sudah diberikan lampu pada *measurement box*, namun cahaya dari luar sedikit banyak masih memberikan pengaruh.

Untuk membantu mengurangi efek dari cahaya itu, digunakan juga fungsi *dilate*, sehingga apabila ada bercak-bercak atau daerah kecil dari pesawat yang tidak terbaca oleh kamera akan tertutupi dan menjadi satu *blob*.

Setelah itu, barulah masuk ke fungsi utama yaitu *findContours*.

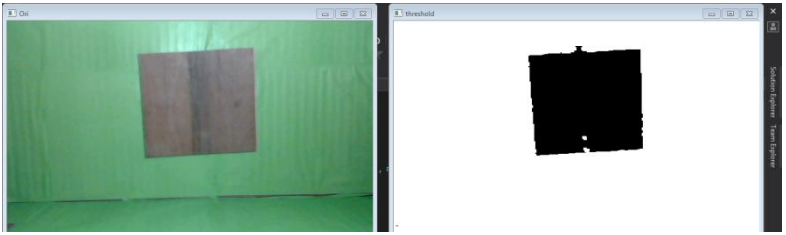
```
findContours(frameTHR, contours, hierarchy,
RETR_TREE, CHAIN_APPROX_NONE, Point(0, 0));
vector<Moments> mu(contours.size());
int wingspan[25];
double wing_area;
for (int i = 0; i < contours.size(); i++)
{
          mu[i] = moments(contours[i], false);
          int area = contourArea(contours[i]);
          if (area > 5000 && area < 200000) {
                    printf("Luas Area = %d\n", area);
          }
}
```

Fungsi *findContours* akan mencari semua *blob* yang ada dalam satu layar. Karena program hanya menginginkan lebar pesawat dan bukan *blob-blob* lain, maka diberikan nilai minimum dan maksimum dari momen *contour*, dimana batas bawah untuk menghindari *noise* dan batas atas untuk menghindari kamera membaca nilai maksimum saat tidak ada pesawat dalam tangkapan kamera atau hanya ada latar belakang dalam tangkapan kamera.

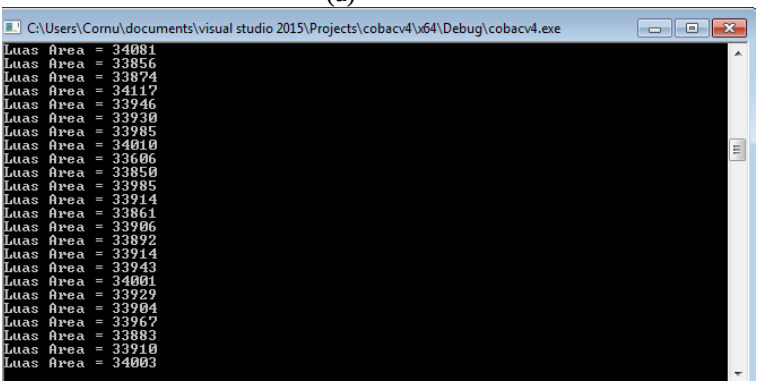
3.5.3. Kalibrasi Skala

Dengan menggunakan program *contour,* maka didapatkan momen dari gambar atau UAV yang ada dalam tangkapan kamera. Namun, satuan dari nilai momen ini masih dalam pixel sehingga dibutuhkan metode agar keluaran dari program ini adalah luasan daerah dalam satuan m^2. Maka dari itu, dilakukanlah kalibrasi skala.

Kalibrasi yang dimaksud disini yaitu mencoba program dengan menggunakan benda yang luasannya telah diketahui. Dalam hal ini, digunakan potongan kayu dengan luas +/- 0.36m x 0.47m atau 0.1692m^2. Potongan kayu ini kemudian dimasukkan ke dalam *measurement box* untuk diketahui luasan pixelnya.



(a)

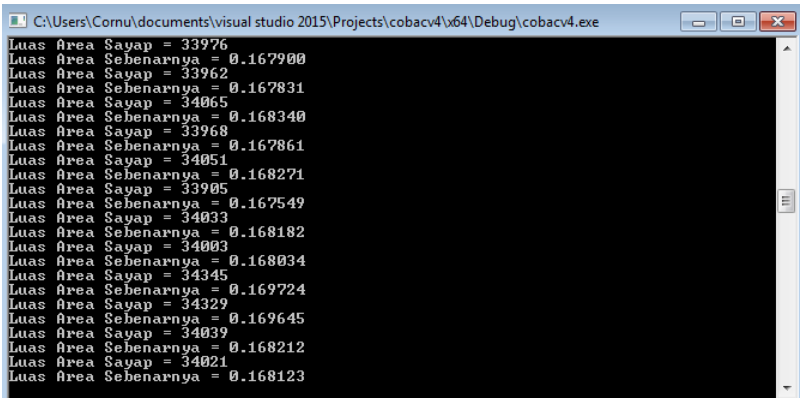


(b)

Gambar 3.21. (a) Hasil tangkapan kamera, (b) Hasil pembacaan luasan pixel

Dengan program yang ada, didapatkan nilai luasan pixel rata-rata sebesar 34000. Menggunakan prinsip sederhana dari perbandingan, maka bisa didapatkan skala dari pixel ke m^2 yaitu $\frac{0.1692}{34000}$.

Skala yang telah didapatkan kemudian dimasukkan ke dalam program dengan sedikit merubah program sebelumnya.



Gambar 3.22. Hasil program setelah kalibrasi skala

```
if (area > 5000 && area < 200000) {
                wing_area = area * 0.1692 / 34000;
                printf("Luas Area = %d\n", area);
                printf("Luas Area Sayap = %f\n", wing_area);
              }
```

Dengan demikian, hasil akhir dari program secara keseluruhan dapat dilihat pada gambar 3.22. Hasil yang dikeluarkan pun tidak berbeda jauh dengan luas potongan kayu yang digunakan, sehingga program ini bisa dibilang sudah cukup akurat.

3.6. Perangkat Pendukung

Untuk menguji coba perangkat lunak, diperlukan perangkat pendukung yang dapat menyimpan data ketinggian dari UAV setiap satuan waktunya agar dapat melakukan komparasi data riil dan data dari hasil perhitungan perangkat lunak. Maka dari itu, dibuatlah perangkat pendukung ini yang terdiri dari NodeMCU, SD *Card Module*, dan BMP180.

Dalam gambaran kasar, perangkat ini akan mendata ketinggian yang akan dibaca melalui BMP180 dan disimpan pada kartu memori dengan modul SD *Card*. Semua itu akan diproses oleh mikrokontroler NodeMCU dengan sumber tegangan dari baterai li-po.

Gambar 3.23. Perangkat pendukung pembaca ketinggian

Tegangan input didapatkan dari baterai li-po setelah melewati regulator untuk merubah tegangan 7.4V li-po menjadi 5V. Perangkat ini memiliki bobot sekitar 0.167 kg, cukup ringan untuk dapat ditempelkan ke UAV sebagai alat bantu *data-logging*.

Mikrokontroler NodeMCU dapat diprogram sama halnya dengan Arduino, sehingga secara tidak langsung juga dapat memanfaatkan *library* yang ada untuk Arduino, termasuk *library* untuk BMP180 dan modul SD *Card*.

Dengan menggunakan *library*, maka fungsi yang digunakan pun relatif sederhana. Untuk bagian *setup*, dilakukan initialisasi dan akan mengembalikan tulisan eror apabila sensor baik BMP180 maupun modul SD *card* tidak terbaca.

```
void setup() {
  Serial.begin(9600);
  if (!bmp.begin()) {
  Serial.println("Could not find a valid BMP085/BMP180
sensor, check wiring!");
  }
  Serial.print("BMP Initialized");
  Serial.print("Initializing SD Card...");
  delay(1000);
  if (!SD.begin(chipSelect)) {
  Serial.println("Error, no SD Card detected.");
```

```
    return;
    }
    Serial.println("Card Initialized.");
}
```

Setelahnya, masuk ke fungsi berulang dimana nantinya tiap satuan waktunya perangkat akan membaca nilai ketinggian dan melakukan *data-logging* dari setiap pembacaannya.

```
void loop() {
    i = i +1;
    x = bmp.readAltitude(alt) - scale;
    if(i==1){
        y = (w * x) + ((1-w)*baseline);
    }
    else{
        y = (w * x) + ((1-w)*z);
    }
```

Karena pembacaan BMP180 cukup sering mengalami *overshoot*, maka untuk menguranginya diberikan program filter sederhana yaitu *exponential filter*.

Filter eksponensial adalah filter yang akan memberikan nilai yang lebih halus dibandingkan tanpa filter, sesuai dengan nilai *weight* yang digunakan. Dalam rumus, dapat dituliskan sebagai berikut :

$$y_n = w \times x_n + (1 - w) \times y_{n-1}$$

dimana y_n adalah keluaran saat n, x_n adalah masukan yang didapatkan saat itu, dan y_{n-1} adalah keluaran filter dari waktu n-1 atau keluaran sebelumnya.[24] Dengan kata lain, keluaran filter adalah penjumlahan antara nilai masukan yang dibaca saat itu dengan nilai keluaran filter sebelumnya dengan pembanding tertentu yaitu *w* atau *weight*. Semakin besar *weight*, maka filter akan lebih mengacu pada nilai masukan yang baru. Sebaliknya, semakin kecil nilai *weight* maka filter akan mengacu lebih banyak pada nilai keluaran sebelumnya.

Meskipun sudah diberi filter, namun data masih mungkin mengalami *noise* sehingga nanti saat data akan digunakan bisa diolah terlebih dahulu.

```
File dataFile = SD.open("LOG.csv", FILE_WRITE);
if(dataFile){
      Serial.println("Writing Data Success.");
      dataFile.print(i); dataFile.print("\t");
      dataFile.print(y); dataFile.print("\t");
      dataFile.print("\n");
      dataFile.close();
}
else {
   Serial.println("Can't write on LOG.csv");
}
Serial.print("Real altitude = ");
Serial.print(y);
Serial.println(" meters");
Serial.println();
y = z;

delay(20);
```

Delay 20ms berarti data akan dicatat setiap 20ms atau 50 kali dalam 1 detik. Semakin banyak data akan semakin membantu untuk melakukan pengecekan terhadap hasil aplikasi.

BAB 4
PENGUJIAN DAN ANALISIS

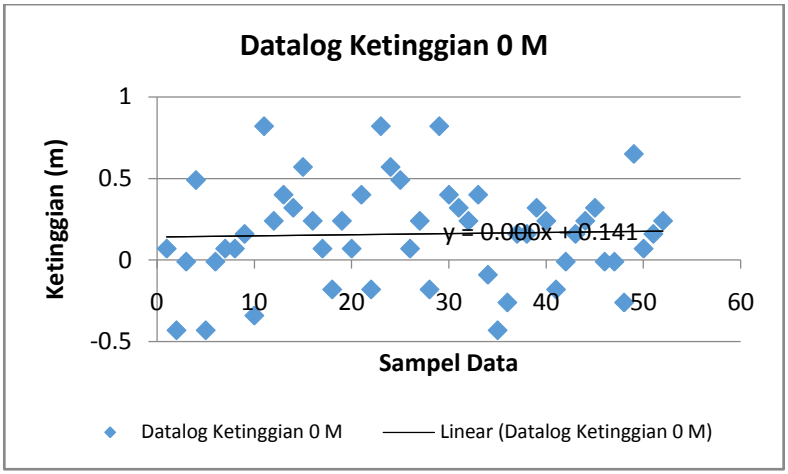
Bab ini akan membahas tentang pengujian dari sistem yang telah dirancang dan dibuat pada bab sebelumnya. Dari hasil pengujian, didapatkan data-data yang akan dibandingkan dengan data riil kemudian dilakukan analisis untuk mengetahui efektifitas sistem baik secara parsial maupun keseluruhan.

4.1. Pengujian Perangkat Pendukung

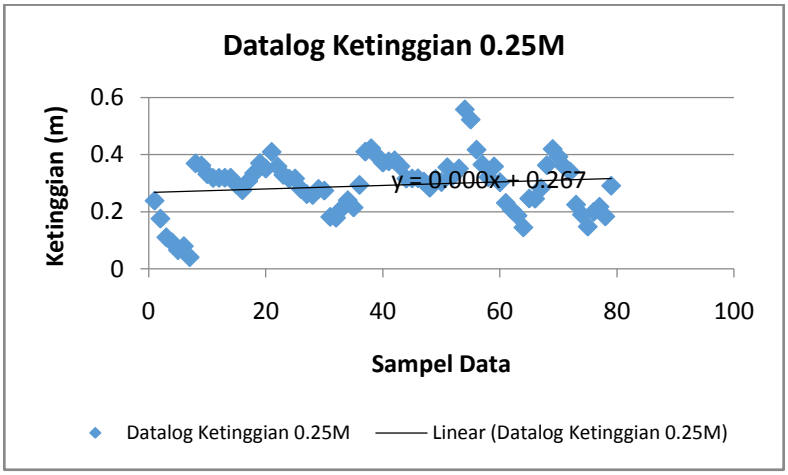
Sensor BMP180 yang sudah dirangkai bersama dengan mikrokontroler dan perangkat lain terpasang pada *board* mikrokontroler akan diuji untuk membaca ketinggian tertentu. Pengujian dilakukan di Laboratorium B202 dengan menggunakan bantuan meteran yang ditempelkan pada tembok. Tujuan dari pengujian ini adalah untuk dapat mengetahui sensitivitas sensor terhadap ketinggian dan mengetahui error dari tiap pembacaan ketinggian. Untuk ketinggian yang diujikan adalah dari 0m hingga 1.5m, dan sebelum pengujian perangkat pendukung melewati proses kalibrasi terlebih dahulu.

Gambar 4.1. Pengujian dengan perangkat pendukung

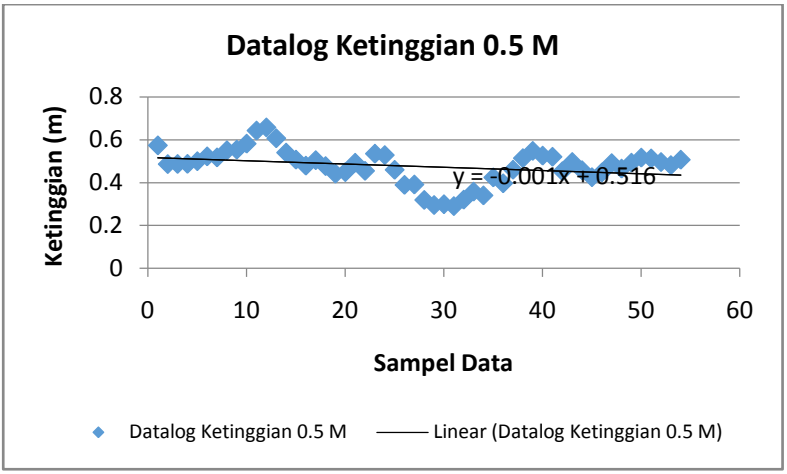
Hasil pembacaan dari perangkat pendukung yang disimpan dalam log di kartu memori kemudian di plot pada setiap pembacaan ketinggiannya. Dikarenakan data masih dalam bentuk pembacaan tiap waktunya, maka untuk mendapatkan data fix dilakukan regresi linier.



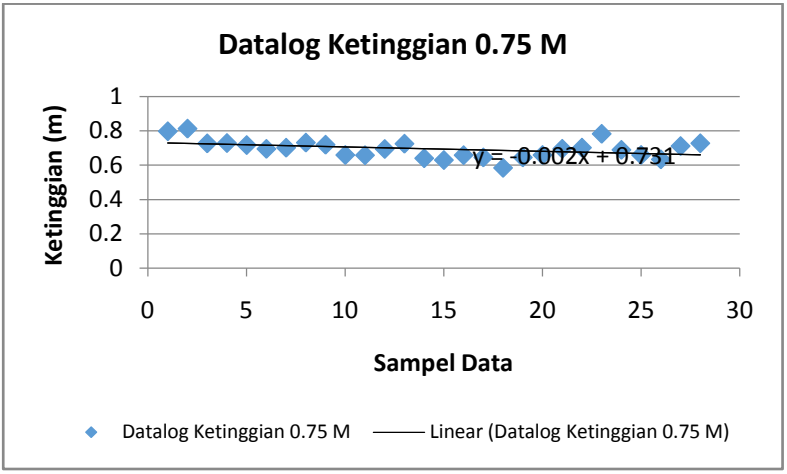
Gambar 4.2. Hasil pembacaan sensor ketinggian 0m



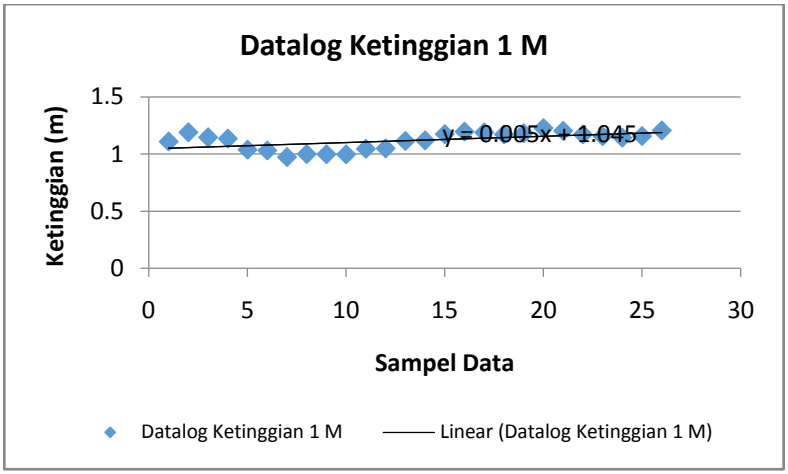
Gambar 4.3. Hasil pembacaan sensor ketinggian 0.25m



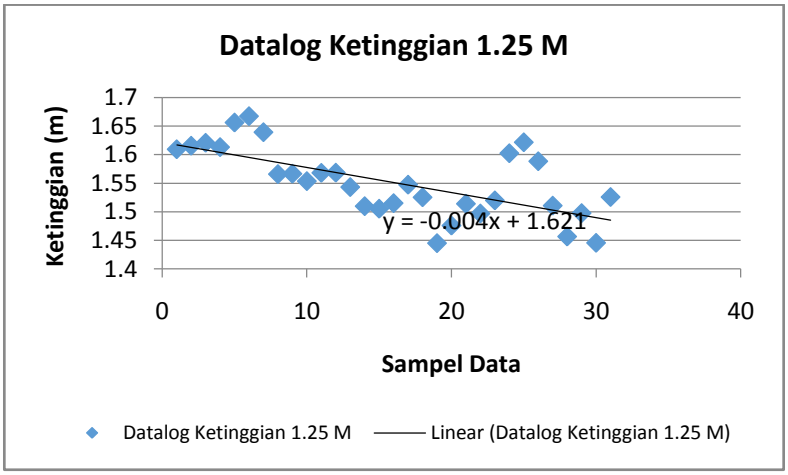
Gambar 4.4. Hasil pembacaan sensor ketinggian 0.5m



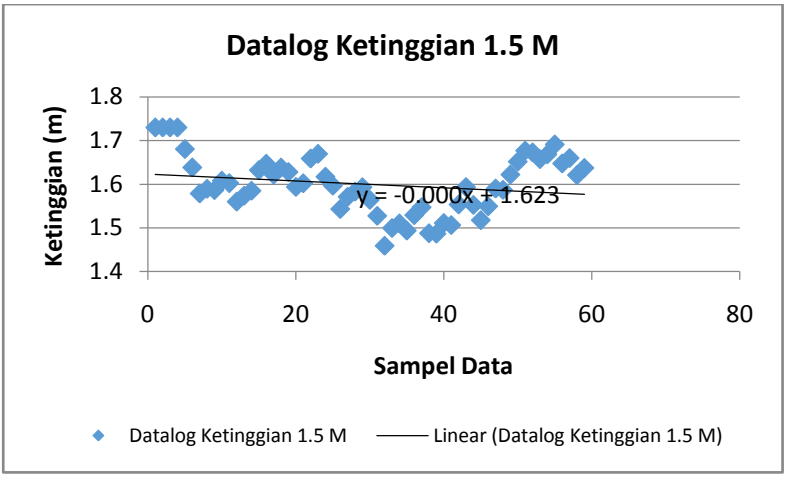
Gambar 4.5. Hasil pembacaan sensor ketinggian 0.75m



Gambar 4.6. Hasil pembacaan sensor ketinggian 1m



Gambar 4.7. Hasil pembacaan sensor ketinggian 1.25m



Gambar 4.8. Hasil pembacaan sensor ketinggian 1.5m

Tabel 4.1. Hasil Pengujian Sensor BMP180

Ketinggian terhadap permukaan (meter)	Pembacaan BMP180 (meter)	Error (meter)
0	0.141	0.141
0.25	0.267	0.017
0.5	0.516	0.016
0.75	0.731	0.019
1	1.045	0.045
1.25	1.621	0.371
1.5	1.623	0.123

Dari data yang didapat, terlihat bahwa pembacaan ketinggian berubah-ubah walau perangkat dalam keadaan diam meskipun telah digunakan filter eksponensial. Regresi linier menghasilkan nilai error yang cukup kecil (kecuali untuk pembacaan 1.25m), namun akan sulit apabila pembacaan tidak stabil seperti itu. Hal ini mungkin saja karena pembacaan sensor BMP180 sangat dipengaruhi oleh tekanan udara, dimana tekanan udara ini yang akan menjadi parameter penentu

pembacaan ketinggian. Oleh karenanya, sangat memungkinkan hal ini terjadi. Maka dari itu, diperlukan kalibrasi yang pas setiap akan menggunakan perangkat ini.

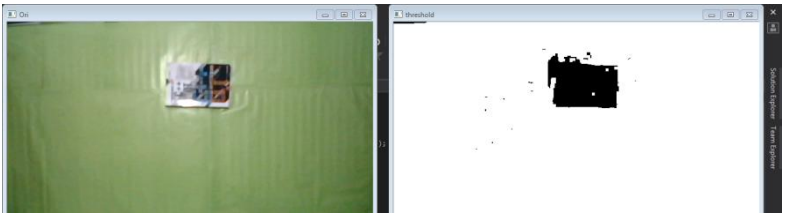
4.2. Pengujian Pembacaan *Wingspan*

Pengujian perangkat keras bertujuan untuk menguji akurasi dari pembacaan *measurement box* untuk luasan benda. Untuk itu, digunakan beberapa objek yang telah diketahui nilai luasannya dan kemudian dibaca menggunakan *measurement box.*

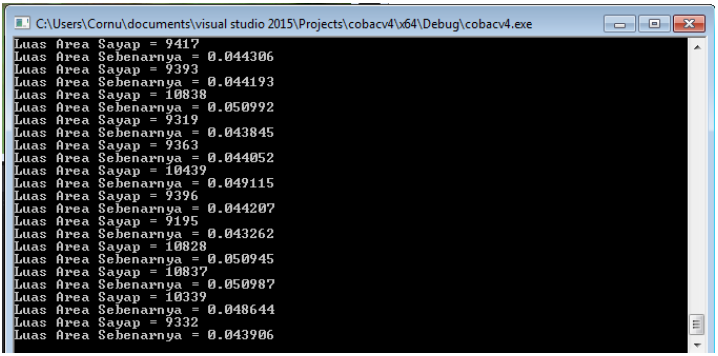
Gambar 4.9. Objek percobaan

Objek yang akan diukur nilai luasannya dimasukkan ke dalam *measurement box* dan aplikasi dijalankan menggunakan MS *Visual Studio* 2015. Untuk pengujian ini, digunakan 6 objek dengan ukuran yang berbeda-beda. Nantinya data luas yang didapatkan dari aplikasi akan dibandingkan dengan ukuran riil yang sudah dimiliki sehingga didapatkan akurasi serta rata-rata akurasi dari keseluruhan pengujian *measurement box* ini. Apabila akurasi terbilang cukup baik, maka *measurement box* dapat dikatakan bekerja dengan baik. Namun sebaliknya apabila eror cukup besar, maka perlu adanya perbaikan serta pengujian ulang.

Hasil dari pengujian dengan objek-objek ini ditunjukkan oleh gambar 4.10 hingga gambar 4.16



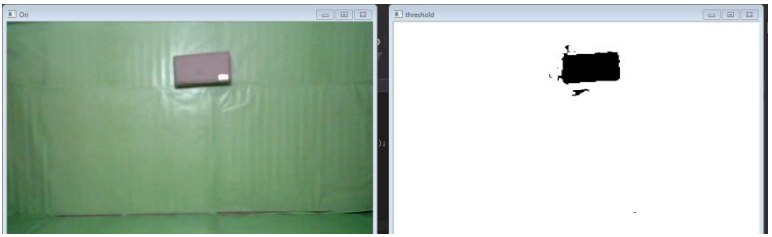
(a)



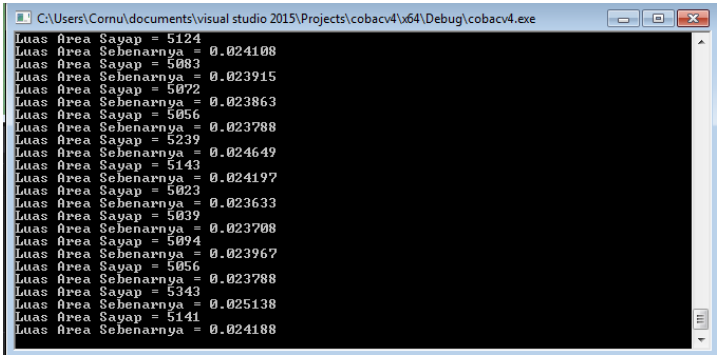
(b)

Gambar 4.10. (a) Pengujian *measurement box* dengan objek 1. (b) Hasil pembacaan

Pembacaan objek pertama terbilang cukup baik terlihat dari *contour* yang didapatkan, hanya saja ada sedikit *noise* dari pantulan cahaya meskipun masih masuk dalam toleransi. Seperti tujuan awal pembuatan *measurement box* ini, warna selain hijau akan dianggap sebagai objek. Namun untuk beberapa kasus seperti pantulan cahaya ini membuat latar belakang hijau yang dimiliki menjadi berwarna lebih terang sehingga terhitung sebagai objek. Hal ini dikarenakan untuk latar belakang warna digunakan kertas warna sehingga masih memiliki efek memantulkan cahaya. Apabila menggunakan kayu yang telah dicat, mungkin akan menghasilkan hasil yang lebih baik.



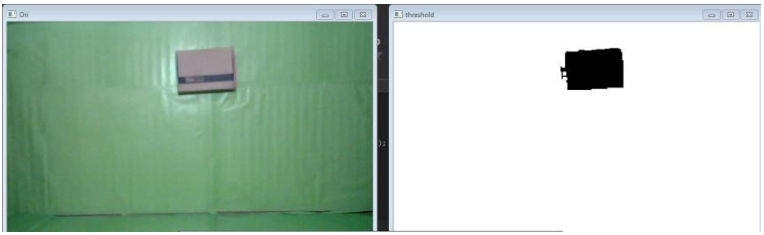
(a)



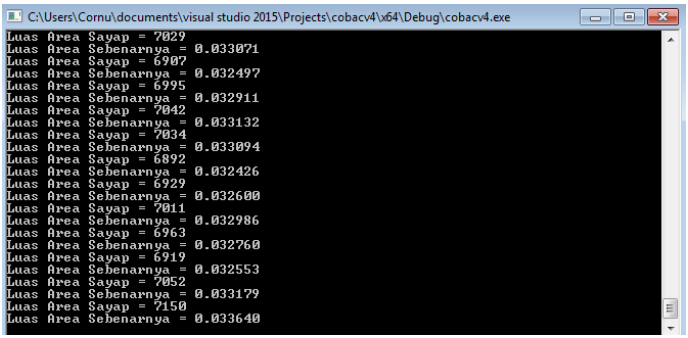
(b)

Gambar 4.11. (a) Pengujian *measurement box* dengan objek 2. (b) Hasil pembacaan

Pengujian dengan objek 2 memiliki *noise* yang lebih sedikit apabila dibandingkan dengan pengujian yang pertama. Dapat dilihat perbandingan antara gambar 4.10a dengan 4.11a bahwa bercak hitam di sekeliling objek lebih sedikit dan bentuk benda lebih menyerupai



(a)



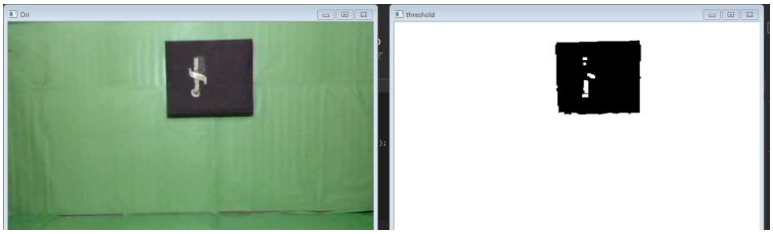
(b)

Gambar 4.12. (a) Pengujian *measurement box* dengan objek 3. (b) Hasil pembacaan

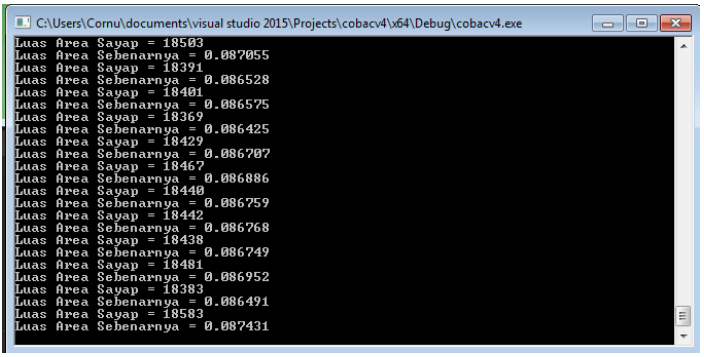
Pembacaan pada objek kedua dan ketiga pada gambar 4.11 dan 4.12 menghasilkan hasil yang lebih baik dikarenakan keduanya merupakan objek yang memiliki warna homogen. Dapat dilihat bahwa hampir tidak ada *noise* dari hasil tangkapan *contour* program OpenCV yang dibuat.

Sejauh ini hasil pembacaan juga menghasilkan pembacaan dengan nilai yang stabil. Artinya, untuk dapat menghasilkan nilai pembacaan sesungguhnya yakni dengan cara merata-rata hasil pembacaan dari tiap sampel data yang diambil tidak memiliki perbedaan yang signifikan.

Pada gambar 4.13 atau pengujian dengan objek 4, terlihat bahwa ada bagian dari objek yang tidak terbaca dikarenakan berwarna putih. Pada kasus ini, bagian yang tidak terbaca terbilang cukup kecil sehingga tidak berpengaruh banyak – hal ini mungkin saja terjadi



(a)

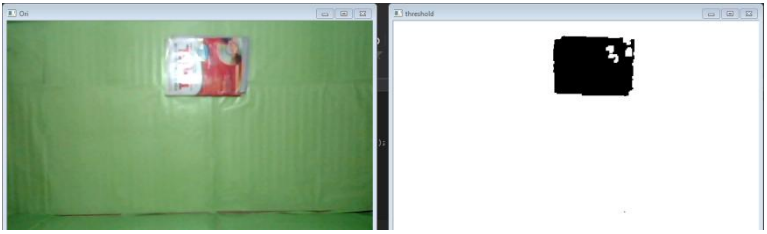


(b)

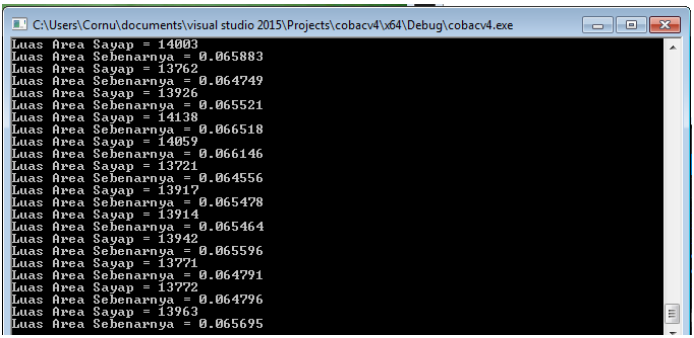
Gambar 4.13. (a) Pengujian *measurement box* dengan objek 4. (b) Hasil pembacaan

. Untuk mencegah hal ini terjadi, dalam program telah digunakan fungsi *dilate* yaitu memperluas pixel *contour* yang terbaca sehingga meminimalisir adanya bagian dalam *contour* yang tidak terbaca. Akan sulit nantinya apabila objek yang dibaca memiliki warna yang lebih banyak dan lebih dominan warna hijau atau putih.

Hal yang sama juga terjadi pada objek 5, dimana pada objek 5 ini terdapat bagian dari benda yang tidak terbaca atau berwarna putih. Namun demikian, bagian yang tidak terbaca tersebut tidaklah signifikan sehingga tidak banyak mempengaruhi hasil.



(a)

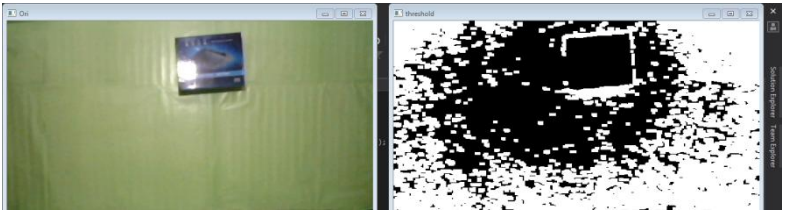


(b)

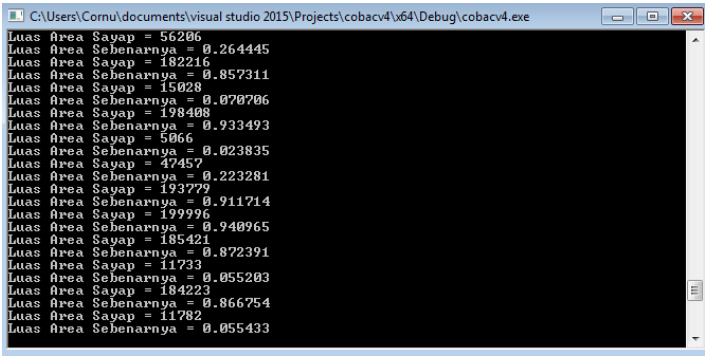
Gambar 4.14. (a) Pengujian *measurement box* dengan objek 5. (b) Hasil pembacaan

Pada pengujian objek 6, terlihat bahwa hasil tangkapan *contour* tidak hanya objek namun juga latar belakang masuk di dalamnya. Terlihat pada gambar 4.15b bahwa data yang didapat sangat banyak dan berubah-ubah, sangat tidak stabil apabila dibandingkan dengan pengujian sebelumnya sehingga pengguna tidak dapat mengetahui mana yang merupakan *contour* dari objek sesungguhnya.

Apabila dibandingkan dengan hasil tangkapan kamera sebelumnya, dapat dilihat secara kasat mata bahwa hijau yang ditangkap oleh pengujian ini sangat berbeda dengan hijau pada pengujian sebelum-sebelumnya. Hal ini dapat terjadi karena terlihat bahwa objek memantulkan cahaya. Cahaya yang dipantulkan objek mengenai kamera sehingga terjadi efek bias yang menyebabkan adanya perbedaan *value* dari warna yang ditangkap oleh kamera.



(a)

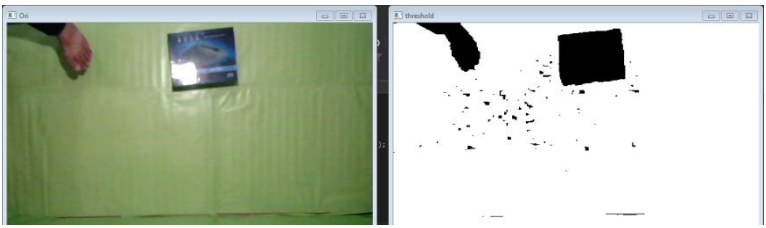


(b)

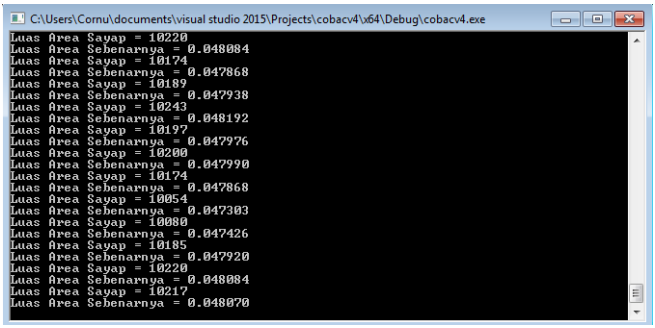
Gambar 4.15. (a) Pengujian *measurement box* dengan objek 6. (b) Hasil pembacaan

Hal ini menyebabkan pecahnya gambar dan *contour* yang didapat pun terpecah. Uniknya, apabila diberikan objek tambahan, maka *contour* yang didapat cenderung stabil dan tidak pecah kembali. Hal ini bisa saja terjadi karena objek tambahan yang dimaksud disini adalah kaki, dan saat memasukkan kaki ke dalam area tangkapan kamera, pengguna menutupi bagian samping dari *measurement box* sehingga efek cahaya dari luar bisa dikurangi. Hasil pembacaan dari objek 6 ini pun terbilang cukup baik pada kasus ini, berkisar ~80% apabila dibandingkan dengan ukuran riil dari benda di dunia nyata.

Dapat dilihat pada gambar 4.16(a) dan (b) bahwa hasil tangkapan kamera terlihat lebih baik.



(a)



Gambar 4.16. (a) Pengujian *Measurement box* dengan objek 6 dengan tambahan objek. (b) Hasil pembacaan

Hasil pengujian-pengujian secara keseluruhan kemudian dimasukkan ke dalam tabel 4.2. dan didapatkan rata-rata akurasi dari *measurement box* ini.

Tabel 4.2. Pengujian *measurement box* dengan beberapa objek

Objek	Dimensi (m x m)	Luas Permukaan Riil (m^2)	Luas Permukaan Pembacaan (m^2)	Akurasi (%)
1	0.158 x 0.248	0.039184	0.044052	87.58
2	0.104 x 0.23	0.02392	0.023967	99.80
3	0.147 x 0.213	0.31311	0.032600	95.88
4	0.28 x 0.325	0.091	0.086886	95.47
5	0.207 x 0.289	0.059823	0.064796	91.68
6	0.186 x 0.236	0.043896	X	0

Dari hasil pengujian, didapatkan bahwa akurasi pembacaan luas *measurement box* berada pada kisaran 94.08% jika menghiraukan

percobaan 6 yang gagal atau 78.4% jika mengikutkan percobaan 6 yang gagal. Dari hasil pengujian ini pun dapat dikatakan bahwa *measurement box* yang dibuat masih belum efektif terhadap pengaruh pantulan cahaya dari objek serta adanya efek cahaya dari luar.

4.3. Pengujian Aplikasi

Pengujian aplikasi dilakukan dengan menggunakan UAV *dummy*. Dalam pengujian aplikasi ini, ada dua hal yang akan dicek, yaitu kesesuaian *mapping point* atau lintasan dan juga akurasi untuk dapat membaca titik terendah dari lintasan. Apabila rumus *mapping point* memiliki error yang kecil, maka bisa dikatakan rumus dapat digunakan.

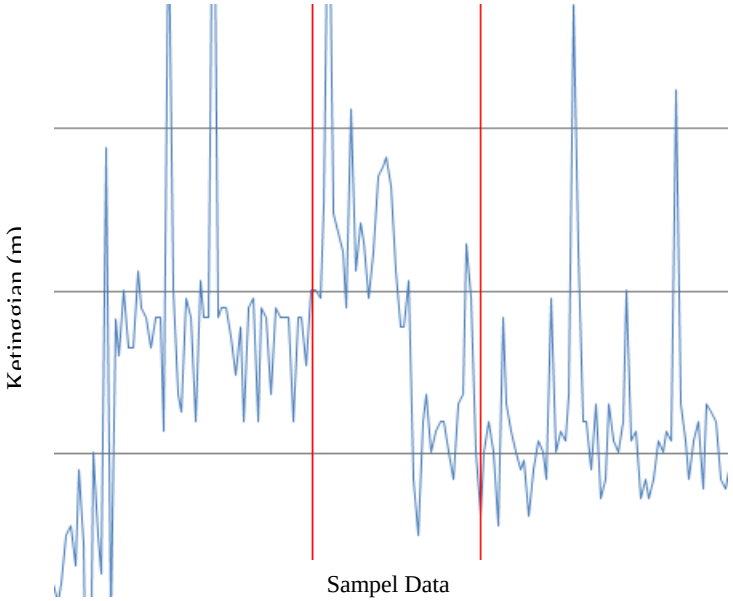
Pada pengujian dengan UAV *dummy*, dilakukan tes dengan beberapa nilai *tension*. Karena tidak memiliki motor atau daya internal, maka *dummy* ini pasti akan jatuh, sehingga titik terendah lintasan adalah titik jatuhnya. Oleh karenanya, perangkat *mobile* tidak dilakukan uji coba karena sudah pasti indikatornya mengarah ke kondisi jatuh.

Tabel 4.3. Pengujian aplikasi dengan UAV *dummy* dengan nilai *tension* yang berbeda

Tension (kg)	Titik Jatuh Aplikasi (m)	Titik Jatuh Riil (m)	Error (m)
20	1.836	1.7	0.136
23.4	2.1981	1.92	0.2781
27	2.5838	2.6	0.0162

Dari tiap pengujiannya itu kemudian dilakukan pencocokan data *mapping point* tiap satuan waktunya, yaitu setiap 20ms. Pencocokan data dilakukan tepat saat UAV lepas landas, atau pada saat UAV mencapai titik tertinggi dari pembacaan sensor BMP180.

Agar mempermudah perbandingan data, maka dari data aplikasi cukup diambil nilai kecepatan dari tiap *tension*, kemudian dilakukan penghitungan ketinggian sumbu *y* dari tiap satuan waktu (baik aplikasi maupun BMP sama-sama melakukan *data-logging* tiap 0.02s).

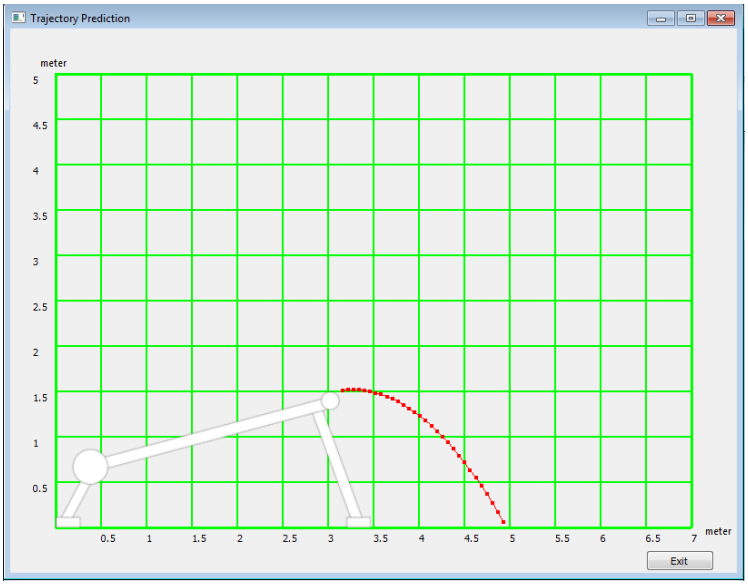


Gambar 4.17. Hasil *data-logging* ketinggian peluncuran dengan *tension* 20 kg

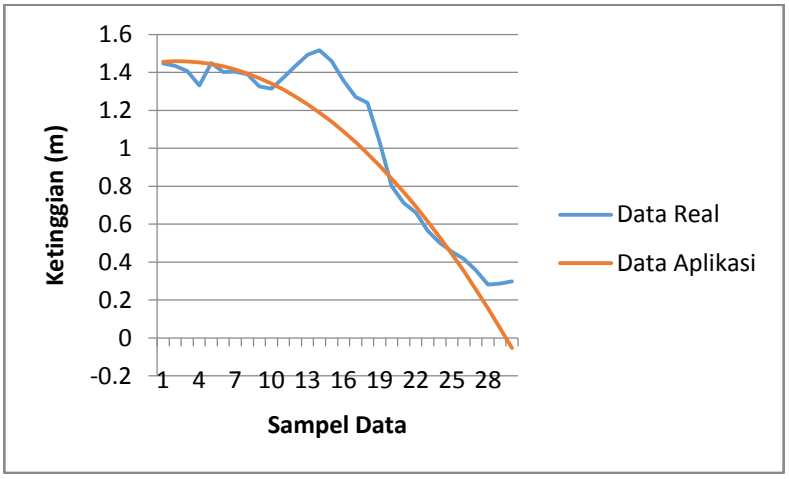
Data yang didapat dari sensor BMP180 cukup kasar meskipun sudah melewati filter eksponensial. Dari hasil pembacaan yang cukup banyak, diambil bagian yang sekiranya merupakan saat-saat UAV lepas landas, cenderung terlihat secara jelas dibanding data lain.

Dapat dilihat pada gambar 4.17. bahwa data dari peluncuran adalah saat ketinggian tiba-tiba merambat naik dan kemudian turun secara pelan-pelan. Memang banyak *overshoot* yang terjadi, namun dengan melakukan pengolahan data dengan menggukana filter. eksponensial dua kali maka data bisa menjadi lebih baik.

Perbandingan antara data riil dengan data aplikasi bisa dibilang ada perbedaan, namun secara kasar dikarenakan pembacaan data BMP180 yang mengalami cukup banyak *noise*. Namun secara umum, kedua data bisa dikatakan sesuai. Hal ini juga didukung dengan rata-rata error dari keseluruhan prediksi lintasan dengan nilai yang tidak terlalu besar, yaitu 0.1154m.



Gambar 4.18. Hasil aplikasi dengan *tension* 20kg

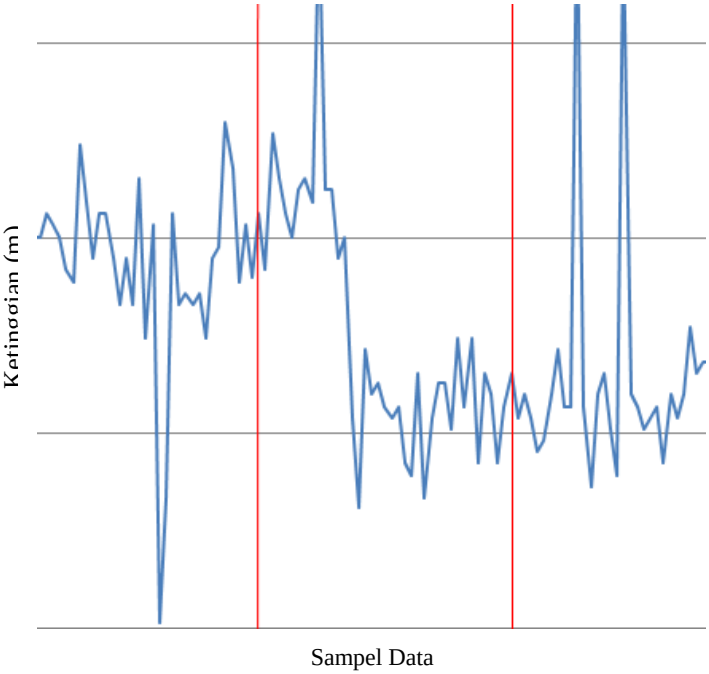


Gambar 4.19. Grafik perbandingan mapping point aplikasi dengan data riil pada pengujian dengan *tension* 20 kg

Tabel 4.4. Perbandingan data ketinggian dengan *tension* 20 kg

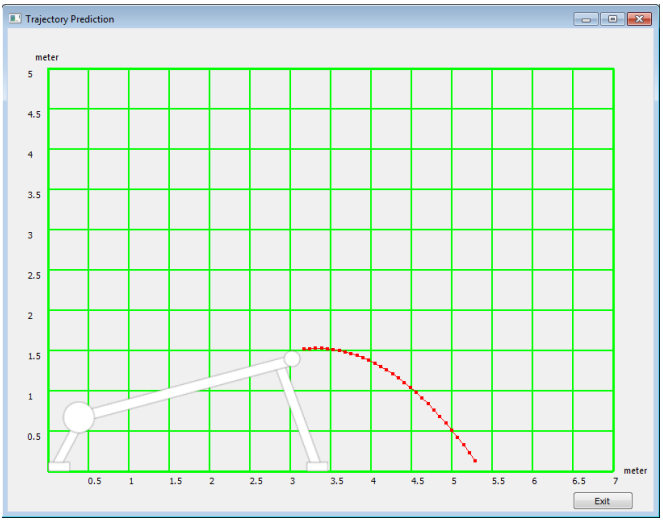
Waktu (s)	Data Riil (m)	Data Aplikasi (m)	Eror (m)
0.02	1.446	1.456792	0.010792
0.04	1.4331	1.459659	0.026559
0.06	1.405635	1.458603	0.052968
0.08	1.32979	1.453622	0.123833
0.10	1.448321	1.444718	0.003603
0.12	1.400573	1.431889	0.031316
0.14	1.403487	1.415137	0.01165
0.16	1.389464	1.394461	0.004996
0.18	1.325044	1.36986	0.044816
0.20	1.313788	1.341336	0.027548
0.22	1.37322	1.308887	0.064332
0.24	1.432737	1.272515	0.160222
0.26	1.492326	1.232218	0.260108
0.28	1.515977	1.187998	0.327979
0.30	1.458081	1.139853	0.318227
0.32	1.356369	1.087785	0.268584
0.34	1.269913	1.031793	0.238121
0.36	1.239926	0.971876	0.26805
0.38	1.031437	0.908036	0.123402
0.40	0.801722	0.840271	0.03855
0.42	0.711463	0.768583	0.057119
0.44	0.660244	0.69297	0.032726
0.46	0.564207	0.613434	0.049227
0.48	0.500576	0.529974	0.029397
0.50	0.45549	0.442589	0.012901
0.52	0.417166	0.351281	0.065886
0.54	0.357591	0.256048	0.101543
0.56	0.281453	0.156892	0.124561
0.58	0.285735	0.053811	0.231923
0.6	0.298375	-0.05319	0.351568
Rata-rata Kesalahan			0.1154

Kemudian dilakukan juga hal yang sama untuk *tension* 23.4 kg dan 27kg, yaitu percobaan kedua dan ketiga dari total tiga percobaan.

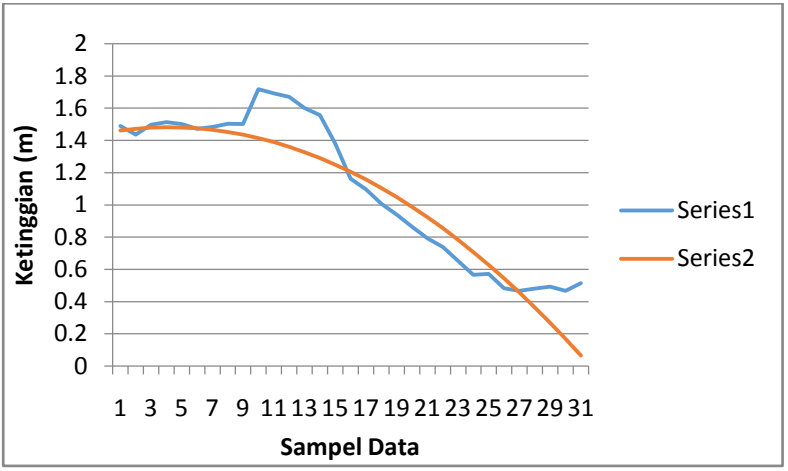


Gambar 4.20. Hasil *data-logging* ketinggian peluncuran dengan *tension* 23.4 kg

Sama halnya dengan penerbangan pertama, data pada penerbangan kedua dengan konfigurasi berbeda ini cenderung memiliki data yang tidak stabil, terlihat dengan banyaknya *overshoot.* Hal ini berbanding terbalik dengan data dari aplikasi yang bisa dilihat pada gambar 4.21. dimana aplikasi ini menganggap bahwa kondisi yang ada adalah kondisi ideal. Kondisi di lapangan, di sisi lain, terpengaruh oleh beberapa hal lain yang menyebabkan baik data dari BMP180 maupun hasil pembacaan secara keseluruhan terlihat berbeda, diantaranya faktor angin dan faktor-faktor mekanis lainnya. Apabila dilakukan percobaan dengan *tension* yang sama pun maka 2 data yang didapat akan berbeda tergantung kondisi lapangan dan kondisi teknis peluncur.



Gambar 4.21. Hasil aplikasi dengan *tension* 23.4 kg

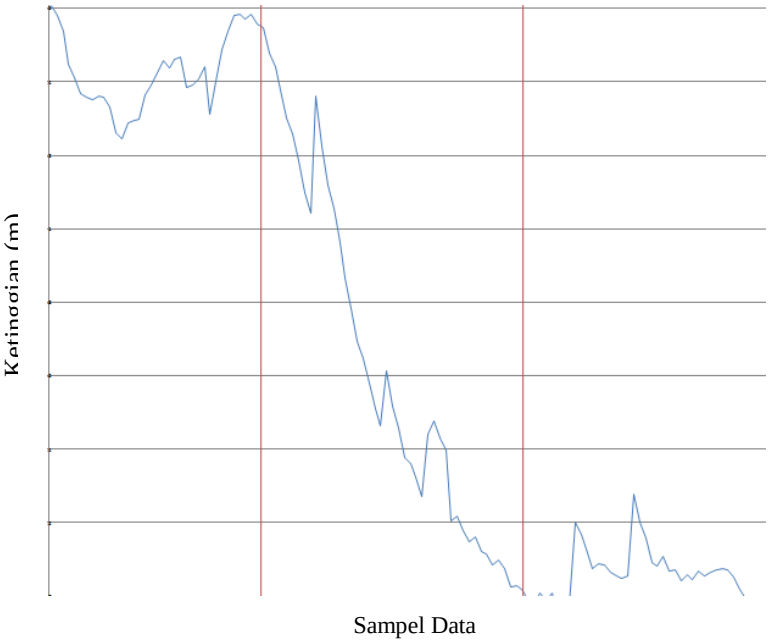


Gambar 4.22. Grafik perbandingan mapping point aplikasi dengan data riil pada pengujian dengan *tension* 23.4 kg

Tabel 4.5. Perbandingan data ketinggian dengan *tension* 23.4 kg

Waktu (s)	Data Riil (m)	Data Aplikasi (m)	Eror (m)
0.02	1.4895	1.46232	0.02718
0.04	1.437075	1.472678	0.035603
0.06	1.497514	1.479112	0.018401
0.08	1.514387	1.481623	0.032764
0.10	1.501729	1.480209	0.02152
0.12	1.472969	1.474871	0.001901
0.14	1.484524	1.465609	0.018915
0.16	1.503345	1.452423	0.050922
0.18	1.501344	1.435313	0.06603
0.20	1.718642	1.414279	0.304363
0.22	1.693346	1.389322	0.304024
0.24	1.671844	1.36044	0.311404
0.26	1.601067	1.327634	0.273433
0.28	1.557407	1.290904	0.266503
0.30	1.380796	1.25025	0.130546
0.32	1.161677	1.205672	0.043996
0.34	1.096925	1.15717	0.060245
0.36	1.007386	1.104745	0.097358
0.38	0.940278	1.048395	0.108116
0.40	0.865237	0.988121	0.122884
0.42	0.792451	0.923923	0.131472
0.44	0.739584	0.855801	0.116218
0.46	0.651146	0.783755	0.132609
0.48	0.566974	0.707785	0.140811
0.50	0.573428	0.627892	0.054464
0.52	0.482914	0.544074	0.06116
0.54	0.467477	0.456332	0.011145
0.56	0.481355	0.364666	0.116689
0.58	0.493152	0.269076	0.224076
0.6	0.467179	0.169562	0.297617
Rata-rata kesalahan			0.1194

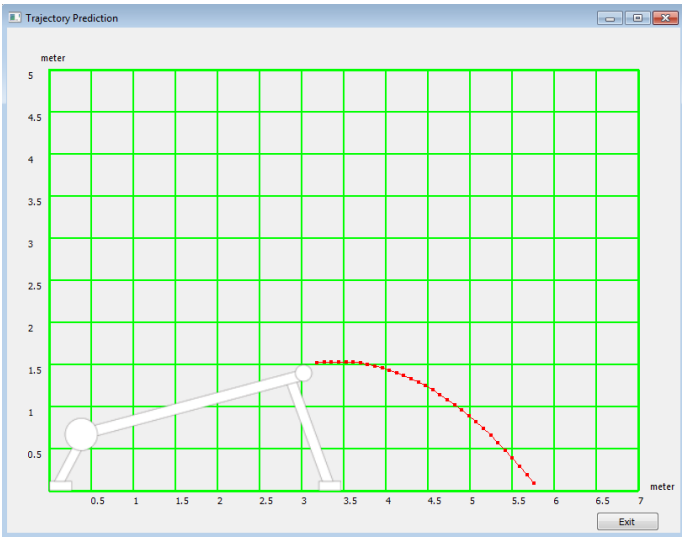
Setelah dilakukan pengolahan data dan perbandingan, eror yang didapat ternyata cukup kecil dan tidak berbeda jauh dengan pengujian pertama (berkisar hanya ~0.1m)



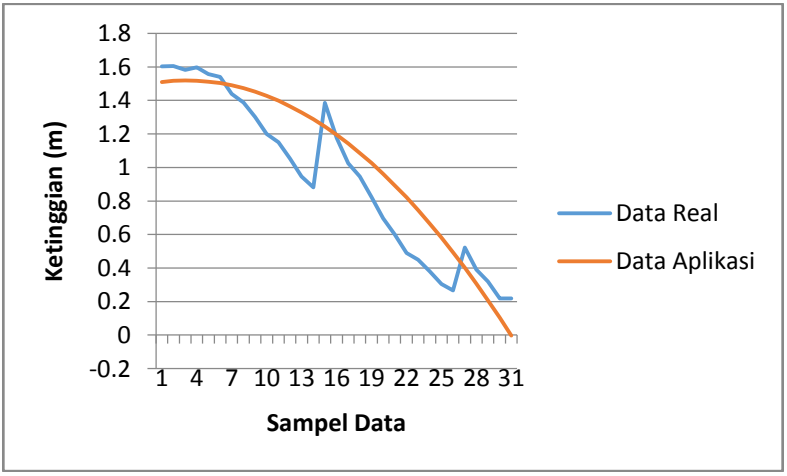
Gambar 4.23. Hasil *data-logging* ketinggian peluncuran dengan *tension* 27 kg

Hasil pembacaan BMP180 pada pengujian ketiga ini bisa dibilang merupakan yang terbaik dibandingkan pada pengujian sebelumnya, meskipun memang masih ada *noise* dan *overshoot*. Seperti yang dijelaskan sebelumnya, faktor lingkungan cukup mempengaruhi hasil pembacaan ketinggian utamanya angin. Dengan data yang cukup bagus ini dapat diasumsikan saat dilakukan pengujian, faktor lingkungan dan faktor mekanis yang ada cukup mendukung.

Dengan data seperti ini, seharusnya tidak perlu dilakukan filter eksponensial kedua karena data sudah cukup stabil. Namun untuk menjaga konsistensi data, data ini tetap harus melewati filter eksponensial dengan nilai *alpha* yang sama, yaitu 0.15.



Gambar 4.24. Hasil aplikasi dengan *tension* 27 kg



Gambar 4.25. Grafik perbandingan mapping point aplikasi dengan data riil pada pengujian dengan *tension* 27 kg

Tabel 4.6. Perbandingan data ketinggian dengan *tension* 27 kg

Waktu (s)	Data Riil (m)	Data Aplikasi (m)	Eror (m)
0.02	1.601518	1.510358	0.09116
0.04	1.60429	1.516792	0.087498
0.06	1.581147	1.519302	0.061844
0.08	1.595975	1.517889	0.078086
0.10	1.556078	1.512551	0.043528
0.12	1.540167	1.503289	0.036878
0.14	1.438142	1.490103	0.051961
0.16	1.38742	1.472993	0.085573
0.18	1.300807	1.451959	0.151152
0.20	1.200186	1.427001	0.226815
0.22	1.149158	1.39812	0.248961
0.24	1.053285	1.365314	0.312029
0.26	0.946292	1.328584	0.382292
0.28	0.880848	1.28793	0.407082
0.30	1.384721	1.243352	0.141369
0.32	1.175513	1.19485	0.019338
0.34	1.023186	1.142424	0.119239
0.36	0.946208	1.086075	0.139867
0.38	0.820777	1.025801	0.205024
0.40	0.69616	0.961603	0.265443
0.42	0.599236	0.893481	0.294245
0.44	0.489851	0.821435	0.331584
0.46	0.449373	0.745465	0.296092
0.48	0.380467	0.665571	0.285104
0.50	0.303897	0.581754	0.277856
0.52	0.265813	0.494012	0.228199
0.54	0.521441	0.402346	0.119095
0.56	0.389225	0.306756	0.082469
0.58	0.320341	0.207242	0.113099
0.6	0.21829	0.103804	0.114485
Rata-rata kesalahan			0.178407

Berdasarkan pengujian dengan *dummy* UAV, maka bisa dikatakan aplikasi telah mampu melakukan *mapping point* posisi ketinggian UAV tanpa faktor motor internal dengan baik (error cukup kecil).

4.4. Pengujian Sistem Keseluruhan

Setelah melakukan pengujian secara parsial, maka langkah terakhir adalah melakukan pengujian sistem secara keseluruhan. Pengujian yang dimaksud adalah menggunakan UAV yang sesungguhnya. Tujuannya adalah untuk dapat membandingkan data riil dengan data aplikasi apakah data yang didapat sudah sesuai atau belum serta mengetahui akurasi dari sistem secara keseluruhan untuk prediksi lintasan yang dibuat.

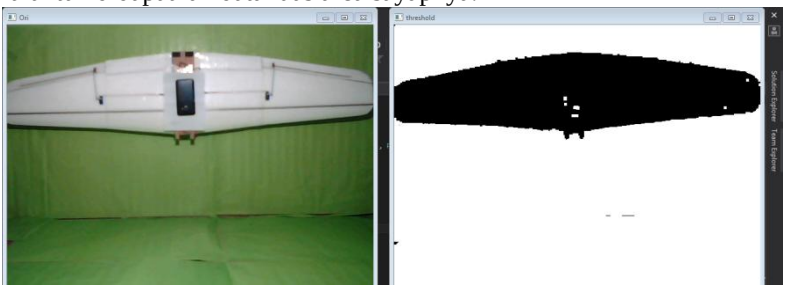
Berikut adalah jenis UAV yang digunakan dan spesifikasinya :

Tabel 4.7. Spesifikasi UAV

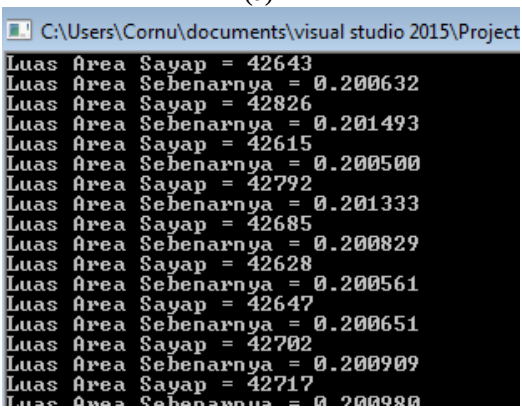
1.	**Dimensi UAV**		
	Panjang sayap	1.4	m
	Panjang badan pesawat	1.1	m
	Berat	1.4	Kg
2.	**Motor dan Propeller**		
	Jumlah motor	1	
	Total power motor	610	Watt
	Jumlah propeler	1	set
	Dimensi propeler	11 x 7	Inch
	Nominal KV	1100	KV

Gambar 4.26. UAV yang digunakan dalam percobaan

Sebelum dilakukan uji coba, maka sama halnya dengan UAV *dummy*, UAV ini dimasukkan ke dalam *Measurement box* terlebih dahulu untuk didapatkan data luas area sayapnya.



(a)



(b)

Gambar 4.27. (a) Hasil pembacaan *Measurement box* (b) Nilai luas sayap UAV

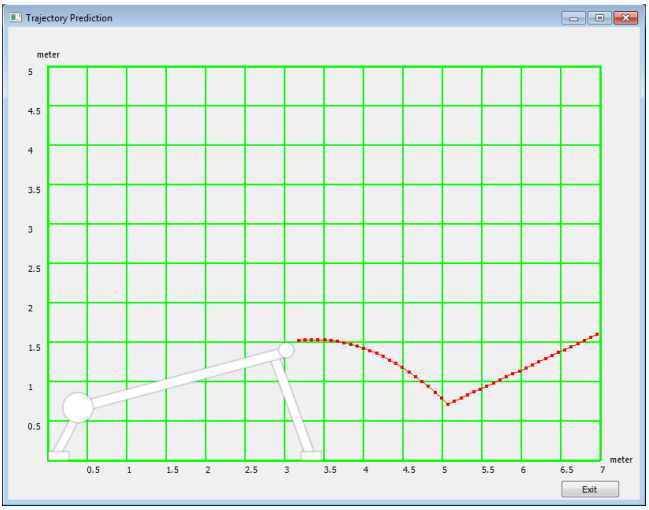
Didapatkan data berupa luas area sayap sebesar +/- 0.2006 m^2. Kemudian didasarkan dari spesifikasi UAV yang ada, maka didapatkan data berupa diameter propeler sebesar 11 inch, *pitch* propeler sebesar 7 inch, serta RPM motor yang di *power-up* menggunakan baterai 3s atau 12v, menjadikan motor memiliki RPM sebesar 13200. Selain itu, juga didapatkan data berat UAV 1.4 kg dan daya motor 610 W. Data-data yang didapat ini akan dimasukkan ke dalam aplikasi untuk mendapatkan prediksi lintasannya.

Pengujian dilakukan di area lapang dekat Gedung Robotika. Sebelum diterbangkan, aplikasi yang telah dibuat dijalankan untuk mendapatkan prediksi lintasan. Pengujian dilakukan sebanyak 3 kali untuk mendapatkan komparasi data yang cukup.

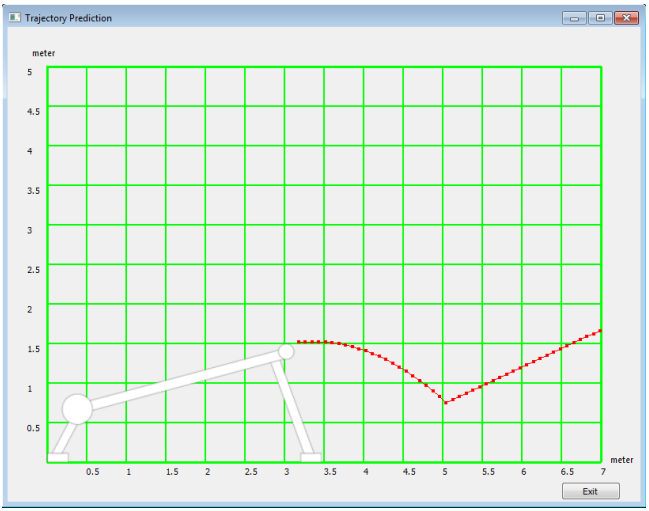
Berbeda dengan UAV *dummy*, UAV ini telah dilengkapi dengan *Pixhawk* yang secara sendirinya melakukan *data-logging* dari ketinggian dari tiap satuan waktunya, meski hanya setiap 0.1s.Data dari penerbangan akan tersimpan dalam bentuk log yang dapat diakses dengan menggunakan aplikasi APM *Planner*.

Dari hasil pengujian penerbangan, akan didapatkan 2 data primer yaitu data kesuksesan penerbangan dan data ketinggian setiap waktunya.

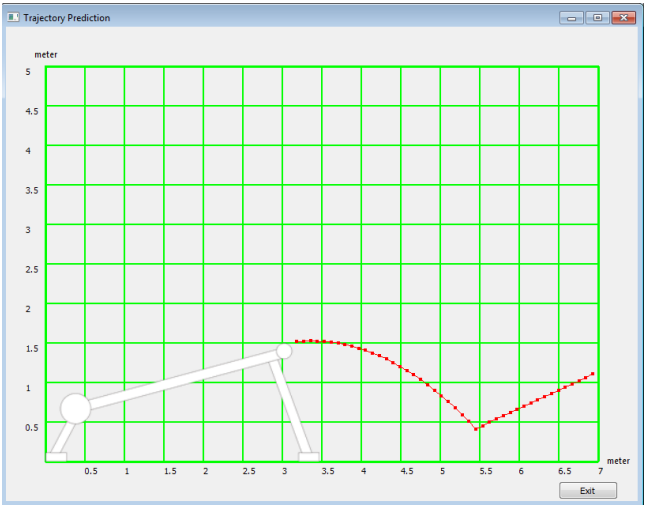
Berikut adalah hasil aplikasi dari 3 penerbangan yang akan dilakukan serta hasil keluaran perangkat *mobile*, dimana perangkat *mobile* hanya akan memberikan keluaran berupa lampu indikator bergantung dari hasil lintasan dan ketinggian *turning point* dari hasil perhitungan aplikasi.



Gambar 4.28. Hasil aplikasi pada penerbangan 2 karet, *tension* 28.4kg dan sudut 9.7º



Gambar 4.29. Hasil aplikasi pada penerbangan 2 karet, *tension* 28.8kg dan sudut 8.4º



Gambar 4.30. Hasil aplikasi pada penerbangan 2 karet, *tension* 30kg dan sudut 8.2º

(a)

(b)

(c)

Gambar 4.31. (a) Keluaran perangkat *mobile* penerbangan pertama. (b) Keluaran perangkat *mobile* penerbangan kedua. (c) Keluaran perangkat *mobile* penerbangan ketiga.

Dapat dilihat dari ketiga hasil aplikasi bahwa UAV akan mampu melewati *turning point* dan dapat sukses terbang, begitu pula dengan perangkat *mobile.* Kemudian dilakukan pengujian sesungguhnya dengan menerbangkan UAV mengikuti konfigurasi peluncur yang sama dengan yang dimasukkan dalam aplikasi.

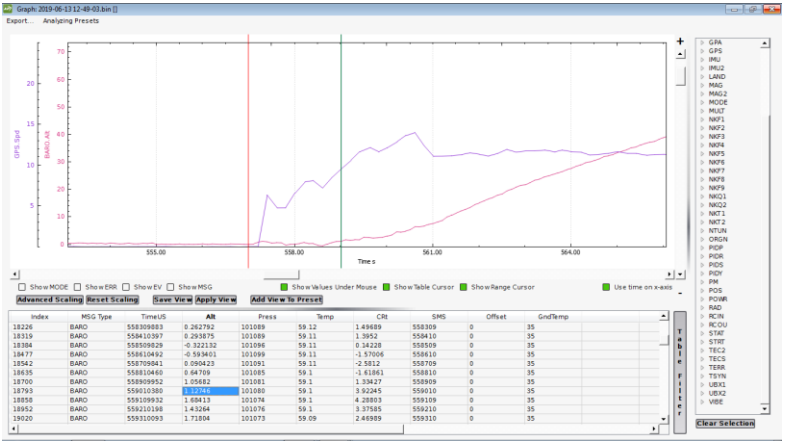
Tabel 4.8. Hasil penerbangan UAV dengan konfigurasi yang berbeda

Percobaan	Jml. Karet	*Tension* (kg)	Sudut (º)	Hasil Penerbangan	Hasil Aplikasi	Hasil Perangkat Mobile
1	2	28.4	9.7	Sukses	Sukses	Sukses
2	2	28.8	8.4	Sukses	Sukses	Sukses
3	2	30	8.2	Sukses	Sukses	Sukses

Dapat dilihat dari segi kesuksesan penerbangan, tidak ada pengujian penerbangan yang gagal terbang dan telah sesuai dengan aplikasi maupun perangkat *mobile*. Dikarenakan pengujian ketinggian tidak bisa dilakukan, maka pengujian perangkat *mobile* hanya bisa dilakukan sebatas berhasil atau tidak, dan tidak bisa membuktikan kebenaran indicator yang dibuat.

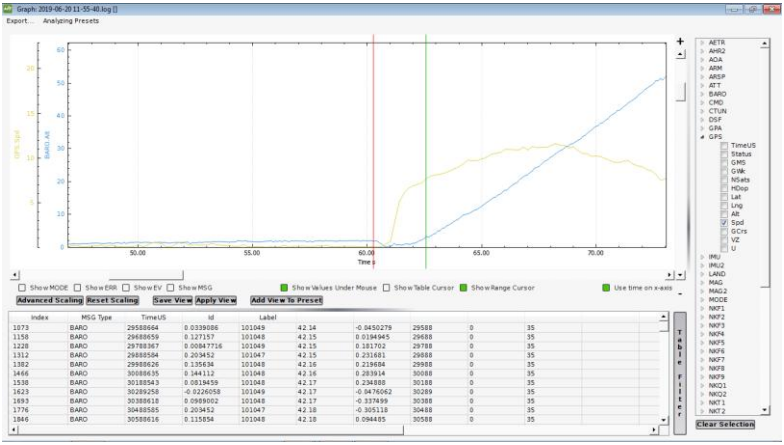
Langkah selanjutnya adalah membandingkan data ketinggian atau *mapping point* dari aplikasi dengan data riil. Berikut adalah pratinjau dari hasil *data-logging* yang didapatkan dari UAV.

Perlu diketahui, data yang ditampilkan disini adalah kecepatan dan ketinggian, sehingga dapat dilihat perbedaan kecepatan saat perubahan ketinggian untuk dapat mengetahui kapan peluncuran pada data yang ada dilakukan.



Gambar 4.32. Datalog *pixhawk* ketinggian dan kecepatan penerbangan pertama

Gambar 4.33. Datalog *pixhawk* ketinggian dan kecepatan penerbangan kedua

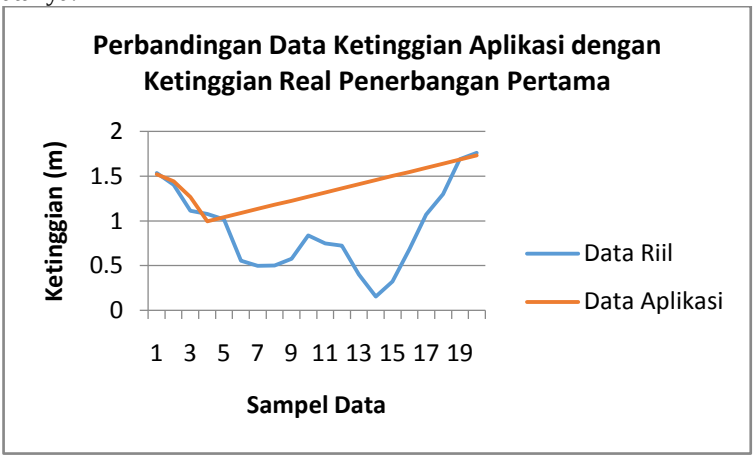


Gambar 4.34. Datalog *pixhawk* ketinggian dan kecepatan penerbangan ketiga

Dari data yang didapatkan, dapat dilihat bahwa proses peluncuran terjadi bersamaan dengan adanya kenaikan nilai kecepatan. Data ketinggian dari proses tersebut kemudian diambil, diolah dengan

menggunakan filter eksponensial dan kemudian dibandingkan dengan data yang didapatkan dari aplikasi. Data yang di *highlight* adalah *turning point* dari lintasan terkait.

Berikut akan ditampilkan grafik perbandingan data aplikasi dengan data riil beserta tabel perbandingan data dari tiap waktu pengambilan datanya.



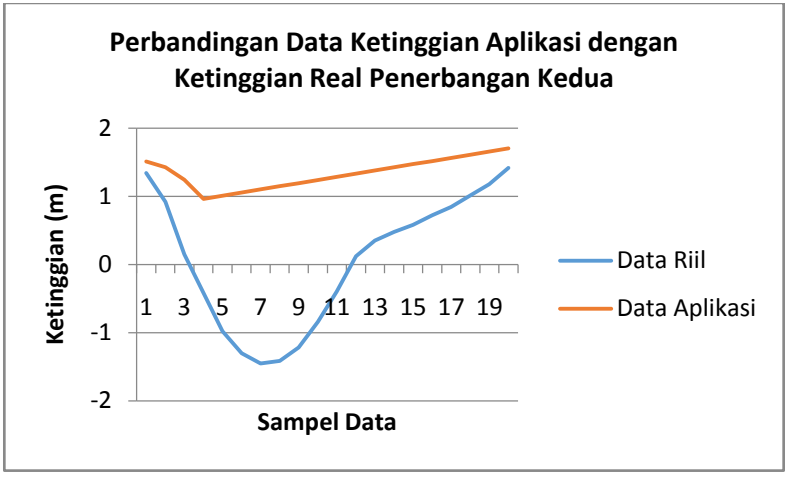
Gambar 4.35. Grafik perbandingan ketinggian data riil dan prediksi penerbangan pertama

Tabel 4.9. Perbandingan data ketinggian penerbangan pertama

Waktu (s)	Data Riil (m)	Data Aplikasi (m)	Eror (m)
0.1	1.536	1.5213	0.0147
0.2	1.403	1.4445	0.0415
0.3	1.1135	1.2696	0.1561
0.4	1.07625	0.9966	0.07965
0.5	1.017625	1.042564	0.024939
0.6	0.5533125	1.088528	0.535216
0.7	0.49615625	1.134493	0.638336
0.8	0.501578125	1.180457	0.678878
0.9	0.575289063	1.226421	0.651132
1	0.837644531	1.272385	0.43474
1.1	0.749822266	1.318349	0.568527
1.2	0.721911133	1.364313	0.642402
1.3	0.399955566	1.410277	1.010322

Waktu (s)	Data Riil (m)	Data Aplikasi (m)	Eror (m)
1.4	0.153477783	1.456241	1.302763
1.5	0.322238892	1.502205	1.179966
1.6	0.684619446	1.548169	0.86355
1.7	1.070809723	1.594134	0.523324
1.8	1.298904861	1.640098	0.341193
1.9	1.691452431	1.686062	0.005391
2	1.761726215	1.732026	0.0297

Pada penerbangan pertama, dapat dilihat terjadi selisih 0.3s antara waktu *turning point* hasil prediksi lintasan dengan waktu *turning point* yang terjadi di percobaan langsung. Eror ketinggian pun cukup tinggi, berkisar kurang lebih 0.45m. Dikarenakan perhitungan tidak memperhitungkan efek *drag* dan juga karena faktor barometer yang mungkin kurang stabil, maka hal ini sangat mungkin terjadi.



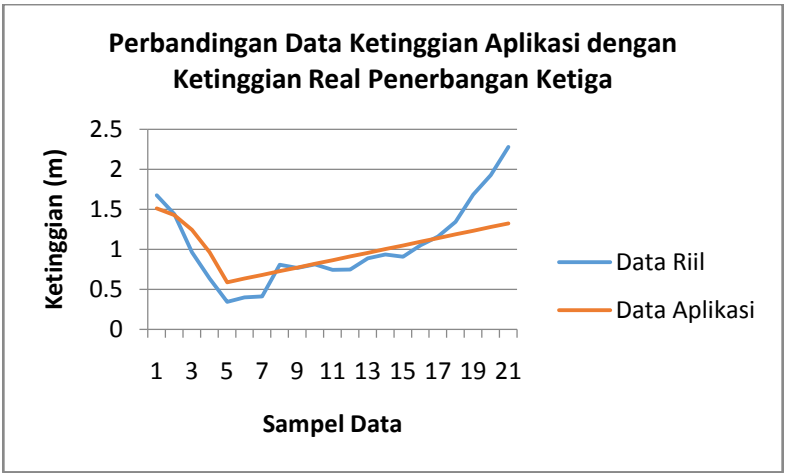
Gambar 4.36. Grafik perbandingan ketinggian data riil dan prediksi penerbangan kedua

Tabel 4.10. Perbandingan data ketinggian penerbangan kedua

Waktu (s)	Data Riil (m)	Data Aplikasi (m)	Eror (m)
0.1	1.344	1.513536	0.169536
0.2	0.918	1.428973	0.510973
0.3	0.1485	1.246309	1.097809
0.4	-0.42025	0.965546	1.385796
0.5	-0.978625	1.011677	1.990302
0.6	-1.3003125	1.057807	2.35812
0.7	-1.45265625	1.103938	2.556594
0.8	-1.41582813	1.150068	2.565896
0.9	-1.21641406	1.196199	2.412613
1	-0.84320703	1.242329	2.085536
1.1	-0.39610352	1.28846	1.684563
1.2	0.121448242	1.33459	1.213142
1.3	0.348724121	1.380721	1.031997
1.4	0.479362061	1.426851	0.947489
1.5	0.58168103	1.472982	0.891301
1.6	0.723040515	1.519112	0.796072
1.7	0.844520258	1.565243	0.720723
1.8	1.011260129	1.611373	0.600113
1.9	1.176130064	1.657504	0.481374
2	1.417065032	1.703634	0.286569

Cukup disayangkan karena penerbangan kedua mengalami anomali data. Baik dari grafik gambar 4.35 maupun dari tabel 4.10 dapat dilihat bahwa barometer membaca nilai ketinggian hingga nilai minus meskipun sudah melewati filter eksponensial. Dikarenakan hal ini, maka penerbangan kedua terbilang tidak relevan apabila dijadikan referensi sebagai hasil pengujian.

Selanjutnya dilakukan pengujian terbang yang ketiga dengan konfigurasi peluncur yang berbeda. Namun, kali ini pada UAV diberi beban tambahan yaitu perangkat pendukung yang memiliki massa +/- 0.1682kg, menjadikan total massa UAV yang digunakan sebesar +/- 1.5682kg. Dapat dilihat bahwa *turning point* dengan konfigurasi peluncur yang lebih tinggi justru menghasilkan ketinggian yang lebih rendah. Hal ini karena perbedaan massa yang ada pada UAV yang digunakan.



Gambar 4.37. Grafik perbandingan ketinggian data riil dan prediksi penerbangan ketiga

Tabel 4.11. Perbandingan data ketinggian penerbangan ketiga

Waktu (s)	Data Riil (m)	Data Aplikasi (m)	Eror (m)
0.1	1.676	1.513781	0.162219
0.2	1.44	1.429463	0.010537
0.3	0.964	1.247044	0.283044
0.4	0.6345	0.966526	0.332026
0.5	0.34255	0.587907	0.245357
0.6	0.398275	0.634061	0.235786
0.7	0.4096375	0.680215	0.270578
0.8	0.80631875	0.726369	0.079949
0.9	0.766159375	0.772523	0.006364
1	0.811079688	0.818677	0.007598
1.1	0.741539844	0.864831	0.123292
1.2	0.747769922	0.910986	0.163216
1.3	0.887884961	0.95714	0.069255
1.4	0.93544248	1.003294	0.067851
1.5	0.90572124	1.049448	0.143726
1.6	1.05286062	1.095602	0.042741
1.7	1.15793031	1.141756	0.016175
1.8	1.344965155	1.18791	0.157056

Waktu (s)	Data Riil (m)	Data Aplikasi (m)	Eror (m)
1.9	1.683982578	1.234064	0.449919
2	1.926991289	1.280218	0.646774

Jika mengesampingkan data dari penerbangan kedua, maka dapat dilihat pada data penerbangan pertama dan ketiga bahwa aplikasi dapat memetakan posisi ketinggian dengan baik hingga titik *turning point*, meski ada eror dari waktu dan posisi dari titik *turning point* itu sendiri.

Tabel 4.12. Hasil akhir aplikasi

Percobaan	*T.P. Error* (s)	*T.P. Error* (m)	*Trajectory Error before T.P.* (m)	*Trajectory Error overall* (m)
1	0.3	0.541	0.246	0.5109
3	0	0.2358	0.2115	0.2127
Rata-rata	0.15	0.3884	0.2287	0.3618

Dengan hasil pengujian, didapatkan rata-rata error prediksi lintasan sebelum *turning point* sebesar 0.2287m dan rata-rata error keseluruhan prediksi lintasan sebesar 0.3618m. Hasil prediksi lintasan terlihat lebih baik dibandingkan keseluruhan karena memang prediksi lintasan setelah *turning point* hanyalah sekedar asumsi bahwa lintasan yang dilewati merupakan lintasan yang linier dengan sudut kenaikan tertentu.

Error dari waktu dan ketinggian dari *turning point*, disisi lain, bisa terjadi karena prediksi lintasan ini tidak memperhitungkan faktor udara atau *air resistance* dan *drag force*.

BAB 5
PENUTUP

5.1. Kesimpulan

Berdasarkan hasil penelitian dan percobaan yang telah dilakukan pada tugas akhir ini dapat ditarik beberapa kesimpulan sebagai berikut:

a. *Measurement box* yang dirancang masih kurang efektif untuk mengatasi masalah cahaya dan pantulan cahaya dari objek.
b. Aplikasi prediksi lintasan ini mampu memetakan titik ketinggian dengan berbagai macam parameter yang diberikan, utamanya dengan nilai *tension*, sudut, jumlah karet dan berat UAV yang berbeda dengan rata-rata error 0.3618m.
c. Beberapa faktor mekanis dari alat peluncur UAV serta faktor eksternal mempengaruhi keakuratan sistem secara keseluruhan, semisal efek *jerking* dari UAV karena *cradle* yang bergoncang saat proses peluncuran serta efek angin.

5.2. Saran

Dari hasil percobaan Pengembangan Fitur Prediksi Lintasan untuk Peluncur UAV Otomatis, maka ada beberapa poin dari penulis yang bisa dikembangkan lebih lanjut, diantaranya sebagai berikut :

a. Mendesain *measurement box* versi portabel sehingga bisa dibawa kemana-mana bersamaan dengan peluncur UAV, atau mendesain *measurement box* dengan metode selain *findContours* agar lebih akurat
b. Menambahkan pintu depan pada *measurement box* sehingga lebih tahan terhadap efek cahaya dari luar.
c. Memperbaiki *user interface* aplikasi agar lebih menarik dan lebih *user-friendly* serta menambahkan fitur *load and save* agar pengguna tidak perlu banyak-banyak memasukkan data.
d. Memperhitungkan faktor *drag force* agar nilai *thrust* yang digunakan tidak konstan dan dapat lebih dinamis serta menyesuaikan dengan kondisi nyata.

.....Halaman ini sengaja dikosongkan.....

DAFTAR PUSTAKA

[1] P. Guruge, B. B. Kocer, and E. Kayacan, "A novel automatic UAV launcher design by using bluetooth low energy integrated electromagnetic releasing system," in *2015 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, Cebu City, Philippines, 2015, pp. 1–4.

[2] R. Austin, *Unmanned Aircraft Systems: UAVS Design, Development and Deployment*. Chichester, UK: John Wiley & Sons, Ltd, 2010.

[3] Z. Novakovi, Z. Vasi, I. Ili, N. Medar, and D. Stevanovi, "Integration of Tactical - Medium Range UAV and Catapult Launch System," p. 7.

[4] P. D. Nugroho, *Rancang Bangun Bungee Cord Unmanned Aerial Vehicle (UAV) Launcher Otomatis*. Surabaya, 2018.

[5] Z. Novakovi and N. Medar, "Analysis of a UAV Bungee Cord Launching Device," p. 7.

[6] U. C. Yayli *et al.*, "Design optimization of a fixed wing aircraft," *Adv. Aircr. Spacecr. Sci.*, vol. 4, no. 1, pp. 65–80, Jan. 2017.

[7] "Lift Formula." [Online]. Available: https://www.grc.nasa.gov/WWW/K-12/WindTunnel/Activities/lift_formula.html. [Accessed: 23-Jun-2019].

[8] J. Francis, "Launch System for Unmanned Aerial Vehicles for use on RAN Patrol Boats," p. 29, 2010.

[9] D. S. Palupi, Suharyanto, and Karyono, *Fisika untuk SMA dan MA Kelas XI*. Jakarta: Pusat Perbukuan, Departemen Pendidikan Nasional, 2009.

[10] Karyono and D. S. Palupi, *Fisika untuk SMA dan MA Kelas X*. Jakarta: Pusat Perbukuan, Departemen Pendidikan Nasional, 2009.

[11] J. D. Anderson, "Fundamentals of Aerodynamics," p. 1131, 2009.

[12] R. Budi, *Pemrograman GUI Dengan C++ dan QT*. Bandung: Informatika Bandung.

[13] A. Kaehler and G. Bradski, *Learning OpenCV 3*. United States of America: O'Reilly Media, 2016.

[14] S. Subair and L. Abraham, "Intelligent Pressure Measuring System," p. 6, 2014.

[15] M. B. Satheesh, B. Senthilkumar, T. Veeramanikandasamy, O. M. Saravanakumar, and P. Student, "Microcontroller and SD Card Based Standalone Data Logging System using SPI and I2C Protocols for Industrial Application," vol. 5, no. 4, p. 8.

[16] A. A. Dahoud and M. Fezari, “NodeMCU V3 For Fast IoT Application Development,” p. 14.
[17] J. Muliadi, “An empirical method for the catapult performance assessment of the BPPT-developed UAVs,” *J. Phys. Conf. Ser.*, vol. 1130, p. 012033, Nov. 2018.
[18] Z. Novakovi, N. Medar, and L. Mitrovi, “Increasing Launch Capability of a UAV Bungee Catapult,” p. 10.
[19] M. F. B. Jamaludin, M. A. Wahid, M. N. M. Nasir, N. Othman, and MohdZarhamdyMd.Zaid, “Design and Analysis Performance of Fixed Wing VTOL UAV,” *28 Novemb. 2018*, Jun. 2018.
[20] B. Gupta, A. Chaube, A. Negi, and U. Goel, “Study on Object Detection using Open CV - Python,” *Int. J. Comput. Appl.*, vol. 162, no. 8, pp. 17–21, Mar. 2017.
[21] A. Jakubowski, A. Kubacki, B. Minorowicz, and A. Nowak, “Analysis Thrust for Different Kind of Propellers,” in *Progress in Automation, Robotics and Measuring Techniques*, vol. 350, R. Szewczyk, C. Zieliński, and M. Kaliczyńska, Eds. Cham: Springer International Publishing, 2015, pp. 85–90.
[22] “Calculating thrust? - RC Groups.” [Online]. Available: https://www.rcgroups.com/forums/showthread.php?413706-Calculating-thrust. [Accessed: 22-Jun-2019].
[23] “Propeller Static & Dynamic Thrust Calculation,” *Flite Test*. [Online]. Available: //flitetest.com/articles/propeller-static-dynamic-thrust-calculation. [Accessed: 22-Jun-2019].
[24] “Exponential filter for smoothing noisy data | Reference | MegunoLink.” [Online]. Available: https://www.megunolink.com/documentation/arduino-libraries/exponential-filter/. [Accessed: 24-Jun-2019].

LAMPIRAN A

Program QT Creator Pengembangan Peluncur UAV Otomatis dengan Prediksi Lintasan

a. mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QWidget>
#include <QLabel>
#include <QLineEdit>
#include <QFrame>
#include <QPushButton>
#include <secondwindow.h>

class MainWindow : public QWidget
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    void setupUi();
    void calculate();
    void calculate2();
    double v_output;
    double acc_output;
    QLineEdit *line3; // angle
    QLineEdit *line4; // mass
    QLineEdit *line5; // velocity
    QLineEdit *line_b; // wingspan
    QLineEdit *line_e; // acc2

public slots:
    void on_showButton_clicked(){
        this->calculate2();
    }
```

```
    void on_calculateButton_clicked(){
        this->calculate();
    }

private:
    QPushButton *showButton;
    QPushButton *calculateButton;
    SecondWindow *secondWindow;
    QLabel *label1;
    QLabel *label2;
    QLabel *label2a;
    QLabel *label3;
    QLabel *label4;
    QLabel *label5;
    QLabel *label6;
    QLabel *label7;
    QLabel *label8;
    QLabel *label8a;
    QLabel *label9;
    QLabel *label10;
    QLabel *label11;
    QLabel *label12;
    QLabel *label13;
    QLineEdit *line1;
    QLineEdit *line2;
    QLineEdit *line2a;
    QLineEdit *line6;
    QLineEdit *line7;
    QLineEdit *line8;
    QLineEdit *line8a;
    QLineEdit *line9;
    QLineEdit *line10;
    QLineEdit *line11;
    QLineEdit *line12;
    QLineEdit *line13;
    QFrame *verticalLine;
    QLabel *label_a;
    QLabel *label_a1;
    QLabel *label_b;
```

```
    QLabel *label_c;
    QLabel *label_d;
    QLabel *label_e;
    QLabel *label_f;
    QLineEdit *line_a;
    QLineEdit *line_a1;
    QLineEdit *line_c;
    QLineEdit *line_d;
    QLineEdit *line_f;

};

#endif // MAINWINDOW_H
```

b. mainwindow.cpp

```
#include "mainwindow.h"
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QPalette>

MainWindow::MainWindow(QWidget *parent) :
    QWidget(parent)
{
    this->setupUi();
    secondWindow = new SecondWindow();

    connect(line5,SIGNAL(textChanged(QString)),
            secondWindow,SLOT(receiveData(QString)));

    connect(line3,SIGNAL(textChanged(QString)),
            secondWindow,SLOT(receiveAngle(QString)));

    connect(line4,SIGNAL(textChanged(QString)),
            secondWindow,SLOT(receiveMass(QString)));

    connect(line_b,SIGNAL(textChanged(QString)),
            secondWindow,SLOT(receiveWingspan(QString)));
```

```
	connect(line_e,SIGNAL(textChanged(QString)),
			secondWindow,SLOT(receiveAcc(QString)));
}

MainWindow::~MainWindow()
{
	delete showButton;
	delete secondWindow;
	delete label1;
	delete label2;
	delete label3;
	delete label4;
	delete label5;
	delete label6;
	delete label7;
	delete label8;
	delete label9;
	delete label10;
	delete label11;
	delete line1;
	delete line2;
	delete line3;
	delete line4;
	delete line5;
	delete line6;
	delete line7;
	delete line8;
	delete line9;
	delete line10;
	delete line11;
}

void MainWindow::setupUi(){
	this->setWindowTitle("UAV Path Modelling");
	this->resize(800,600);

	QPalette *palette = new QPalette();
	palette->setColor(QPalette::Base,Qt::lightGray);
```

```
palette->setColor(QPalette::Text,Qt::white);

QPalette *palette2 = new QPalette();
palette2->setColor(QPalette::Base,Qt::blue);
palette2->setColor(QPalette::Text,Qt::white);

label1 = new QLabel();
label1->setText("Cords Usage");
line1 = new QLineEdit();

label2 = new QLabel();
label2->setText("Tension on Controller (kg)");
line2 = new QLineEdit();

label2a = new QLabel();
label2a->setText("Propulsive Force (N)");
line2a = new QLineEdit();
line2a->setReadOnly(true);
line2a->setPalette(*palette);

label3 = new QLabel();
label3->setText("Angle");
line3 = new QLineEdit();

label4 = new QLabel();
label4->setText("UAV Mass (kg)");
line4 = new QLineEdit();

label6 = new QLabel();
label6->setText("Coefficient of Elasticity");
line6 = new QLineEdit();
line6->setText("169.44");
line6->setReadOnly(true);
line6->setPalette(*palette);

label7 = new QLabel();
label7->setText("Initial Cord Length (m)");
line7 = new QLineEdit();
line7->setText("0.7");
```

```
line7->setReadOnly(true);
line7->setPalette(*palette);

label8 = new QLabel();
label8->setText("Cord Elongation (m)");
line8 = new QLineEdit();
line8->setReadOnly(true);
line8->setPalette(*palette);

label8a = new QLabel();
label8a->setText("Elongated Cord Length (m)");
line8a = new QLineEdit();
line8a->setReadOnly(true);
line8a->setPalette(*palette);

label9 = new QLabel();
label9->setText("Cradle Mass (kg)");
line9 = new QLineEdit();
line9->setText("3.919");
line9->setReadOnly(true);
line9->setPalette(*palette);

label10 = new QLabel();
label10->setText("Total Mass (kg)");
line10 = new QLineEdit();
line10->setReadOnly(true);
line10->setPalette(*palette);

label11 = new QLabel();
label11->setText("Coefficient of Friction");
line11 = new QLineEdit();
line11->setText("0.1");
line11->setReadOnly(true);
line11->setPalette(*palette);

label12 = new QLabel();
label12->setText("Gravitational Force");
line12 = new QLineEdit();
line12->setText("9.81");
```

```
line12->setReadOnly(true);
line12->setPalette(*palette);

label13 = new QLabel();
label13->setText("Launching Time (s)");
line13 = new QLineEdit();
line13->setReadOnly(true);
line13->setPalette(*palette2);

label5 = new QLabel();
label5->setText("Result of Velocity Calculation (m/s)");
line5 = new QLineEdit();
line5->setReadOnly(true);
line5->setPalette(*palette2);

calculateButton = new QPushButton("Calculate");

verticalLine = new QFrame();
verticalLine->setFrameShape(QFrame::VLine);
verticalLine->setFrameShadow(QFrame::Sunken);

label_a = new QLabel();
label_a->setText("Prop. Diameter (inch)");
line_a = new QLineEdit();

label_a1 = new QLabel();
label_a1->setText("Prop. Pitch (inch)");
line_a1 = new QLineEdit();

label_b = new QLabel();
label_b->setText("Wingspan (m2)");
line_b = new QLineEdit();

label_c = new QLabel();
label_c->setText("Motor RPM");
line_c = new QLineEdit();

label_d = new QLabel();
label_d->setText("Lift Coefficient");
```

```
line_d = new QLineEdit();
line_d->setText("0.6");
line_d->setReadOnly(true);
line_d->setPalette(*palette);

label_e = new QLabel();
label_e->setText("Acceleration (m/s2)");
line_e = new QLineEdit();
line_e->setReadOnly(true);
line_e->setPalette(*palette2);

label_f = new QLabel();
label_f->setText("Thrust (N)");
line_f = new QLineEdit();
line_f->setReadOnly(true);
line_f->setPalette(*palette2);


showButton = new QPushButton("Generate");

QVBoxLayout *vboxa = new QVBoxLayout();
vboxa->addWidget(label2);
vboxa->addWidget(line2);

QVBoxLayout *vboxb = new QVBoxLayout();
vboxb->addWidget(label2a);
vboxb->addWidget(line2a);

QVBoxLayout *vboxc = new QVBoxLayout();
vboxc->addWidget(label7);
vboxc->addWidget(line7);

QVBoxLayout *vboxd = new QVBoxLayout();
vboxd->addWidget(label8);
vboxd->addWidget(line8);

QVBoxLayout *vboxe = new QVBoxLayout();
vboxe->addWidget(label_a);
vboxe->addWidget(line_a);
```

```
QVBoxLayout *vboxf = new QVBoxLayout();
vboxf->addWidget(label_a1);
vboxf->addWidget(line_a1);

QHBoxLayout *hbox0 = new QHBoxLayout();
hbox0->addLayout(vboxa);
hbox0->addLayout(vboxb);

QHBoxLayout *hbox1 = new QHBoxLayout();
hbox1->addLayout(vboxc);
hbox1->addLayout(vboxd);

QHBoxLayout *hbox2 = new QHBoxLayout();
hbox2->addLayout(vboxe);
hbox2->addLayout(vboxf);

QVBoxLayout *vbox1 = new QVBoxLayout();
vbox1->addWidget(label1);
vbox1->addWidget(line1);
vbox1->addLayout(hbox0);
vbox1->addWidget(label3);
vbox1->addWidget(line3);
vbox1->addWidget(label4);
vbox1->addWidget(line4);
vbox1->addWidget(label6);
vbox1->addWidget(line6);
vbox1->addLayout(hbox1);
vbox1->addWidget(label8a);
vbox1->addWidget(line8a);
vbox1->addWidget(label9);
vbox1->addWidget(line9);
vbox1->addWidget(label10);
vbox1->addWidget(line10);
vbox1->addWidget(label11);
vbox1->addWidget(line11);
vbox1->addWidget(label12);
vbox1->addWidget(line12);
vbox1->addWidget(label13);
```

```
        vbox1->addWidget(line13);
        vbox1->addWidget(label5);
        vbox1->addWidget(line5);
        vbox1->addWidget(calculateButton);
        vbox1->addStretch();

        QVBoxLayout *vbox2 = new QVBoxLayout();
        vbox2->addLayout(hbox2);
        vbox2->addWidget(label_c);
        vbox2->addWidget(line_c);
        vbox2->addWidget(label_b);
        vbox2->addWidget(line_b);
        vbox2->addWidget(label_d);
        vbox2->addWidget(line_d);
        vbox2->addWidget(label_f);
        vbox2->addWidget(line_f);
        vbox2->addWidget(showButton);
        vbox2->addStretch();

        QHBoxLayout *layout = new QHBoxLayout();
        layout->addLayout(vbox1);
        layout->addWidget(verticalLine);
        layout->addLayout(vbox2);

        this->setLayout(layout);

        connect(calculateButton,SIGNAL(clicked()),
                this,SLOT(on_calculateButton_clicked()));

        connect(showButton,SIGNAL(clicked()),
                this,SLOT(on_showButton_clicked()));

}

void MainWindow::calculate(){
        double cords = line1->text().toDouble(); //raw
        double tension = line2->text().toDouble(); //raw
        double angle = line3->text().toDouble(); //raw
        double mass_uav = line4->text().toDouble(); //raw
```

```
        double length_elong = line8->text().toDouble(); //calculated
        double mass_total = line10->text().toDouble(); //calculated
        double mass_crad = 3.919;
        double elasticity = 167.4;
        double length_init = 0.7;
        double length_stretch = 0.0;
        double friction = 0.1;
        double gravity = 9.81;
        double velocity = 0.0;
        double time = 0.0;
        double nr = 0.0;
        double phi = 3.1415;
        double force = 0.0;

        force = tension * gravity;

        mass_total = mass_uav + mass_crad;

        length_elong = (force/cords)/ elasticity ;
        length_stretch = length_init + length_elong;

        time = (sqrt(mass_total/elasticity))*acos(((length_init)-

((mass_total*gravity/elasticity)*(friction*cos(angle*phi/180)+

sin(angle*phi/180)))-length_init)/(length_stretch-((mass_total*gravity/el
asticity)*

(friction*cos(angle*phi/180)+sin(angle*phi/180)))-length_init));

        velocity = (length_stretch -

((mass_total*gravity/elasticity)*(friction*cos(angle*phi/180)+sin(angle
*phi/180)))-

length_init)*(sqrt(elasticity/mass_total))*(sin((sqrt(elasticity/mass_total)
)*time));
```

```
    QString s0,s1,s2,s3,s4,s5;
    line2a->setText(s0.setNum(force));
    line8->setText(s1.setNum(length_elong));
    line8a->setText(s2.setNum(length_stretch));
    line10->setText(s3.setNum(mass_total));
    line5->setText(s4.setNum(velocity));
    line13->setText(s5.setNum(time));

    v_output = velocity;
}


void MainWindow::calculate2()
{
    double diameter = line_a->text().toDouble();
    double pitch = line_a1->text().toDouble();
    double rpm = line_c->text().toDouble();
    double mass_uav = line4->text().toDouble(); //raw
    double force2 = (1.225 *
((3.14*(0.0254*diameter)*(0.0254*diameter))/4) *
(rpm*0.0254*pitch/60)*(rpm*0.0254*pitch/60)
*(sqrt((diameter/(3.29547*pitch))*(diameter/(3.29547*pitch))*(diamete
r/(3.29547*pitch)))));
    double acc2 = 0;
    if(force2>0){
        acc2 = force2 / mass_uav;
    }
    QString s0,s1;
    line_e->setText(s0.setNum(acc2));
    line_f->setText(s1.setNum(force2));
    acc_output = acc2;
    secondWindow->show();
}
```

c. secondwindow.h

```
    #ifndef SECONDWINDOW_H
    #define SECONDWINDOW_H
```

```
#include <QWidget>
#include <QPushButton>
#include <QPainter>
#include <QLabel>
#include <QPaintEvent>
#include <QLineEdit>

class SecondWindow: public QWidget{
    Q_OBJECT

public:
    SecondWindow();
    ~SecondWindow();
    void setupUi();
    int height_max = 600;
    int width_max = 800;
    int height_diag = height_max - 100;
    int width_diag = width_max - 100;
    double acc2 = 0;
    double v_out2, mass2, angle2, wingspan2, force2;
    double liftcoeff2 = 0.5;
    QLineEdit *vout;
    QLineEdit *accout;
    QLineEdit *rpm;
    QLineEdit *mass;
    QLineEdit *angle;
    QLineEdit *wingspan;

public slots:
    void receiveData(QString data){
        vout = new QLineEdit();
        vout->setText(data);
        QString s1;
        v_out2 = vout->text().toDouble();
    }
    void receiveMass(QString data){
        mass = new QLineEdit();
        mass->setText(data);
        QString s1;
```

```
        mass2 = mass->text().toDouble();
    }
    void receiveAngle(QString data){
        angle = new QLineEdit();
        angle->setText(data);
        QString s1;
        angle2 = angle->text().toDouble();
    }
    void receiveAcc(QString data){
        accout = new QLineEdit();
        accout->setText(data);
        QString s1;
        acc2 = accout->text().toDouble();
    }
    void receiveWingspan(QString data){
        wingspan = new QLineEdit();
        wingspan->setText(data);
        QString s1;
        wingspan2 = wingspan->text().toDouble();
    }

private slots:
    void on_closeButton_clicked();

protected:
    void paintEvent(QPaintEvent *drw);

private:
    QPushButton *closeButton;
    QLabel *labelx;
    QLabel *labely;
    QLabel *labelp;
    QLabel *velocity;

};

#endif // SECONDWINDOW_H
```

d. secondwindow.cpp

```
#include "secondwindow.h"
#include <iostream>
#include <QHBoxLayout>
#include <QPixmap>


SecondWindow::SecondWindow(){
    this->setupUi();

}

SecondWindow::~SecondWindow(){
    delete closeButton;
}

void SecondWindow::setupUi(){
    setFixedSize(width_max,height_max);
    this->setWindowTitle("Trajectory Prediction");
    double i,j;
    for(i=1;i<=width_diag/50;i++){
        j = i * 0.5;
        QPoint p1;
        p1.setX(50+(i*50));
        p1.setY((height_max-45));
        QString b = QString::number(j);
        labelx = new QLabel(this);
        labelx->setText(b);
        labelx->move(p1);
    }
    for(i=1;i<=height_diag/50;i++){
        j = i * 0.5;
        QPoint p1;
        p1.setX(25);
        p1.setY((height_max-50)-(i*50));
        QString b = QString::number(j);
        labelx = new QLabel(this);
        labelx->setText(b);
```

```
            labelx->move(p1);
        }

        QPixmap pixmap= QPixmap(":/files/uav2.png");
        QPoint p2;
        p2.setX(50);
        p2.setY(height_max-200);
        labelp = new QLabel(this);
        labelp->setPixmap(QPixmap(pixmap));
        labelp->move(p2);

        closeButton = new QPushButton("Exit");
        closeButton->move(700,575);
        closeButton->setParent(this);

        connect(closeButton,SIGNAL(clicked()),
                this,SLOT(on_closeButton_clicked()));

}

void SecondWindow::on_closeButton_clicked(){
        this->close();
}

void SecondWindow::paintEvent(QPaintEvent *drw){
        QPainter painter(this);
        QPen rectpen(Qt::green);
        QBrush brush(Qt::green,Qt::NoBrush);
        rectpen.setWidth(3);
        painter.setPen(rectpen);
        QRectF
bg(QPoint(50,50),QPoint((width_max-50),(height_max-50)));
        painter.drawRect(bg);
        painter.fillRect(bg,brush);

        double i,j;
        for(i=1;i<=width_diag/50;i++){
            QPen linepen(Qt::green);
            linepen.setWidth(2);
```

```
        painter.setPen(linepen);
        j = i * 0.5;
        QPoint p1;
        p1.setX(50+(i*50));
        p1.setY((height_max-50));
        QPoint p2;
        p2.setX(p1.x());
        p2.setY(p1.y()-height_diag);
        QLine horiz(p1,p2);
        painter.drawLine(horiz);
}
for(i=1;i<=height_diag/50;i++){
        QPen linepen(Qt::green);
        linepen.setWidth(2);
        painter.setPen(linepen);
        j = i * 0.5;
        QPoint p1;
        p1.setX(50);
        p1.setY((height_max-50)-(i*50));
        QPoint p2;
        p2.setX(p1.x()+width_diag);
        p2.setY(p1.y());
        QLine vertic(p1,p2);
        painter.drawLine(vertic);
}

QPoint start;
start.setX(360);
start.setY(height_max-200);

QPoint x2;
x2.setX(0);
x2.setY(0);

QPoint x1, turning_point;

double x = 0.0, vt = 0.0, xt = 0.0;
double y = 0.0, yt = 0.0;
double t = 0.0;
```

```
        double weight = 0.0;
        double lift = 0.0;
        double density = 1.225;
        double distance = 0.0;
        double const_ang = 25;
        double tan10 = tan(const_ang*3.1415/180);
        double time_turning = 0.0;
        int k = 0;

        for(double i=1;i<=100;i++){
            if(v_out2 >= 1){
            t = 0.02 * i;
            vt = v_out2 * cos(angle2*3.1415/180)*t + (acc2 * t);
            x = v_out2 * cos(angle2*3.1415/180) * t;

            weight = mass2 * 9.81;
            lift = 0.5 * density * (vt * vt) * wingspan2 *
liftcoeff2;

            if(lift<weight){
                y = ((v_out2 * sin(angle2*3.1415/180) * t ) -
(0.5 * 9.81 * t * t));
            }
            else if (lift>weight){
                distance = xt - x;
                y = yt - (tan10 * distance);
                if(k==0){
                    turning_point.setX(xt);
                    turning_point.setY(yt);
                    time_turning = t-0.02;
                }
                k++;
            }

            if(i>1){
                yt = y;
                xt = x;
                x2.setX(x1.x());
                x2.setY(x1.y());
```

```
                }

                x1.setX(start.x()+(x*100));
                x1.setY(start.y()-(y*100));

                if(x<0){
                    break;
                }

                if(x1.x()>750){
                    break;
                }

                if(y+1.45<0){
                    break;
                }

                if(i>1){
                    QPen ballpen(Qt::red);
                    ballpen.setWidth(1);
                    painter.setPen(ballpen);
                    QLine trajectory1(x1,x2);
                    painter.drawLine(trajectory1);
                }

                QPen ballpen(Qt::red);
                ballpen.setWidth(4);
                painter.setPen(ballpen);
                painter.drawPoint(x1);
                }
            }

        }
```

e. main.cpp

```
#include "mainwindow.h"
#include <QApplication>
```

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow form;

    form.show();
    return a.exec();
}
```

LAMPIRAN B

Program Scanning Box Visual Studio dan OpenCV

```
int main() {
	namedWindow("Ori", CV_WINDOW_NORMAL);
	resizeWindow("Ori", Size(640, 480));
	Mat frame, frameHSV, frameTHR;
	Mat element2 = getStructuringElement(MORPH_RECT, Size(5,
5), Point(-1, -1));
	vector<vector<Point>> contours;
	vector<Vec4i>hierarchy;
	VideoCapture cap(1);
	for (;;) {
		cap >> frame;
		cvtColor(frame, frameHSV, CV_BGR2HSV);
		inRange(frameHSV, Scalar(40, 30, 30), Scalar(80, 255, 255),
frameTHR);
		dilate(frameTHR, frameTHR, element2);
		findContours(frameTHR, contours, hierarchy, RETR_TREE,
CHAIN_APPROX_NONE, Point(0, 0));
		vector<Moments> mu(contours.size());
		int wingspan[25];
		double wing_area;
		for (int i = 0; i < contours.size(); i++)
		{
			mu[i] = moments(contours[i], false);
			int area = contourArea(contours[i]);
			if (area > 5000 && area < 200000) {
				wing_area = area * 0.1692 / 34000;
				printf("Luas Area Sayap = %d\n", area);
				printf("Luas Area Sebenarnya = %f\n", wing_area);
			}
		}
		imshow("Ori", frame);
		imshow("threshold", frameTHR);
		if (waitKey(27) >= 0)break;
	}
	return 0;
}
```

.....Halaman ini sengaja dikosongkan.....

LAMPIRAN C

Program NodeMCU Perangkat Pendukung Pembaca Ketinggian

```
#include <SD.h>
#include <Adafruit_BMP085.h>
#include <Wire.h>

Adafruit_BMP085 bmp;
   int i = 0;
   const int chipSelect = 15;
   float baseline = 0;
   float scale = 50;

void setup() {
   Serial.begin(9600);
   if (!bmp.begin()) {
   Serial.println("Could not find a valid BMP085/BMP180 sensor, check
wiring!");
   }
   Serial.print("BMP Initialized");
   Serial.print("Initializing SD Card...");
   delay(1000);
   if (!SD.begin(chipSelect)) {
   Serial.println("Error, no SD Card detected.");
   return;
   }
   Serial.println("Card Initialized.");
}

void loop() {
   i = i +1;
   float alt = bmp.readAltitude(102000);
   float real_alt = alt - scale;
   File dataFile = SD.open("LOG.csv", FILE_WRITE);
   if(dataFile){
         Serial.println("Writing Data Success.");
         dataFile.print(i); dataFile.print("\t");
         dataFile.print(real_alt); dataFile.print("\t");
         dataFile.print("\n");
```

```
        dataFile.close();
    }
    else {
      Serial.println("Can't write on LOG.csv");
    }
    Serial.print("Real altitude = ");
    Serial.print(real_alt);
    Serial.println(" meters");
    Serial.println();

    delay(50);
}
```

LAMPIRAN D

Program Arduino Perangkat Prediksi Lintasan Mobile

```
#include <Key.h>
#include <Keypad.h>

#include <LiquidCrystal.h>

const int rs = 14, en = 15, d4 = 16, d5 = 17, d6 = 18, d7 = 19;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

const byte ROWS = 4; //four rows
const byte COLS = 4; //three columns
char keys[ROWS][COLS] = {
{'1','2','3','A'},
{'4','5','6','B'},
{'7','8','9','C'},
{'*','0','#','.'}
};
byte rowPins[ROWS] = {10, 9, 8, 7}; //connect to the row pinouts of the
kpd
byte colPins[COLS] = {6, 5, 4, 3}; //connect to the column pinouts of
the kpd
int index = 0;
char numbers[20];
float result =0;
int confirm = 0;
int confirm2 = 0;


double length_elong = 0.0; //calculated
      double mass_total = 0.0; //calculated
      double mass_crad = 3.919;
      double elasticity = 167.4;
      double length_init = 0.7;
      double length_stretch = 0.0;
      double friction = 0.1;
      double gravity = 9.81;
```

```
    double velocity = 0.0;
    double tension = 0.0;
    double angle = 0.0;
    double cords = 0.0;
    double mass_uav = 0.0;
    double wingspan = 0.0;
    double time1 = 0.0;
    double phi = 3.1415;
    double force = 0.0;
    double diameter = 0.0;
    double pitch = 0.0;
    double rpm = 0.0;
    double thrust = 0.0;
    double accel = 0.0;
    double weight = 0.0;
    double lift = 0.0;
    double density = 1.225;
    double distance = 0.0;
    double liftcoeff = 0.5;
    double x = 0.0; double vt = 0.0; double xt = 0.0;
    double y = 0.0; double yt = 0.0;
    double t = 0.0;
    double tp_x = 0.0;
    double tp_y = 0.0;
    double tp_t = 0.0;
    double fp_x = 0.0;
    double fp_y = 0.0;
    double fp_t = 0.0;

Keypad kpd = Keypad( makeKeymap(keys), rowPins, colPins, ROWS,
COLS );

void setup() {
  Serial.begin(9600);
  pinMode(11,OUTPUT);
  pinMode(12,OUTPUT);
  pinMode(2,OUTPUT);
  lcd.begin(16,2);
  lcd.print("Trajectory");
```

```
    lcd.setCursor(0,1);
    lcd.print("Prediction");
    delay(2000);
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("AFA 2019");
    lcd.setCursor(0,1);
    lcd.print("TA A206");
    delay(2000);
    lcd.clear();
}

void loop() {
        lcd.setCursor(0,0);
        lcd.print("Press A to");
        lcd.setCursor(0,1);
        lcd.print("proceed");
        while(confirm2!=1){
           GetStarted();
        }
        lcd.clear();
        int condition = 0;
        lcd.setCursor(0,0);

        lcd.print("Tension :");
        while(confirm!=1){
           GetNumbers();
           tension = result;
        }
        confirm = 0;
        Serial.println(tension);

        lcd.clear();
        lcd.print("Angle :");
        while(confirm!=1){
           GetNumbers();
           angle = result;
        }
        Serial.println(angle);
```

```
confirm =0;

lcd.clear();
lcd.print("Cords Usage :");
while(confirm!=1){
   GetNumbers();
   cords = result;
}
Serial.println(cords);
confirm =0;

lcd.clear();
lcd.print("UAV Mass :");
while(confirm!=1){
   GetNumbers();
   mass_uav = result;
}
Serial.println(mass_uav);
confirm =0;

lcd.clear();
lcd.print("Please wait...");
delay(2000);

//1. Obtaining velocity and time given by parameters

force = tension * gravity;

mass_total = mass_uav + mass_crad;

length_elong = (force/cords)/ elasticity ;
length_stretch = length_init + length_elong;

time1 =
(sqrt(mass_total/elasticity))*acos(((length_init)-((mass_total*gravity/ela
sticity)*(friction*cos(angle*phi/180)+sin(angle*phi/180)))-length_init)/(
length_stretch-

((mass_total*gravity/elasticity)*(friction*cos(angle*phi/180)+sin(angle
```

```
*phi/180)))-length_init));

      velocity = (length_stretch
-((mass_total*gravity/elasticity)*(friction*cos(angle*phi/180)+sin(angle
*phi/180)))-length_init)*(sqrt(elasticity/mass_total))*
      (sin((sqrt(elasticity/mass_total))*time1));

      Serial.print("Velocity = ");
      Serial.println(velocity);
      lcd.clear();
      lcd.print("Init. Velocity:");
      lcd.setCursor(0,1);
      lcd.print(velocity);
      delay(2000);

      lcd.clear();
      lcd.print("Prop. Diameter :");
      while(confirm!=1){
         GetNumbers();
         diameter = result;
      }
      Serial.println(diameter);
      confirm =0;

      lcd.clear();
      lcd.print("Prop. Pitch :");
      while(confirm!=1){
         GetNumbers();
         pitch = result;
      }
      Serial.println(pitch);
      confirm =0;

      lcd.clear();
      lcd.print("Motor RPM :");
      while(confirm!=1){
         GetNumbers();
         rpm = result;
      }
```

```
        Serial.println(rpm);
        confirm =0;

        lcd.clear();
        lcd.print("Wingspan :");
        while(confirm!=1){
           GetNumbers();
           wingspan = result;
        }
        Serial.println(wingspan);
        confirm =0;

        lcd.clear();
        lcd.print("Please wait...");
        delay(2000);
        lcd.clear();

        //2. Obtaining acceleration given by calculated thrust

        thrust = (1.225 * ((phi*(0.0254*diameter)*(0.0254*diameter))/4) *
(rpm*0.0254*pitch/60)*(rpm*0.0254*pitch/60)
*(sqrt((diameter/(3.29547*pitch))*(diameter/(3.29547*pitch))*
        (diameter/(3.29547*pitch)))));

        if(thrust>0){
              accel = thrust / mass_uav;
        }

        Serial.print("Thrust = ");
        Serial.println(thrust); Serial.print("\n");
        Serial.print("Accel = ");
        Serial.println(accel); Serial.print("\n");


        //3. Estimating trajectory and turning point

        for(double i=1;i<=100;i++){
              t = 0.02 * i;
              vt = velocity * cos(angle*3.1415/180)*t + (accel * t);
```

```
            x = vt * cos(angle*3.1415/180) * t;

            weight = mass_uav * gravity;
            lift = 0.5 * density * (vt * vt) * wingspan * liftcoeff;

            if(lift<weight){
                  y = ((velocity * sin(angle*phi/180) * t ) - (0.5 * 9.81 * t *
t));
            }

            else if (lift>weight){
                tp_x = xt;
                tp_y = yt + 1.5;
                tp_t = t-0.02;
                if(tp_y > 0.5){
                  condition = 1;
                }
                else{
                  condition = 2;
                }
                break;
            }

            if(i>1){
                yt = y;
                xt = x;
            }

            if(y+1.5<0){
                  fp_x = xt;
                  fp_y = yt + 1.5;
                  fp_t = t-0.02;
                  condition = 3;
                  break;
                  //fall
            }

      }
      Serial.print("Condition=");
```

```
Serial.println(condition); Serial.print("\n");
lcd.print("Condition=");
lcd.setCursor(0,1);
lcd.print(condition);
if(condition==1){
   digitalWrite(12,HIGH);
   lcd.clear();
   lcd.setCursor(0,0);
   lcd.print("TP. Time:");
   lcd.setCursor(10,0);
   lcd.print(tp_t);
   lcd.setCursor(0,1);
   lcd.print("x:");
   lcd.print(tp_x);
   lcd.setCursor(9,1);
   lcd.print("y:");
   lcd.print(tp_y);
}
else if(condition==2){
   lcd.clear();
   lcd.setCursor(0,0);
   lcd.print("TP. Time:");
   lcd.setCursor(10,0);
   lcd.print(tp_t);
   lcd.setCursor(0,1);
   lcd.print("x:");
   lcd.print(tp_x);
   lcd.setCursor(9,1);
   lcd.print("y:");
   lcd.print(tp_y);
   digitalWrite(2,HIGH);
}
else if(condition==3){
   digitalWrite(11,HIGH);
   lcd.clear();
   lcd.setCursor(0,0);
   lcd.print("TP. Time:");
   lcd.setCursor(10,0);
   lcd.print(fp_t);
```

```
      lcd.setCursor(0,1);
      lcd.print("x:");
      lcd.print(fp_x);
    }
    delay(15000);
    digitalWrite(12,LOW);
    digitalWrite(11,LOW);
    digitalWrite(2,LOW);
}

void GetNumbers()
{
     char key = kpd.getKey();
     lcd.setCursor(0,1);
     if(key != NO_KEY)
     {
          if(key == '*')
          {
               index = 0;
               numbers[index] = '\0';
               Serial.println(numbers);
          }
          else if(key == '.')
          {
               numbers[index++] = '.';
               numbers[index] = '\0';
               Serial.println(numbers);
          }
          else if(key >= '0' && key <= '9')
          {
               numbers[index++] = key;
               numbers[index] = '\0';
               Serial.println(numbers);
          }
          else if(key == '#')
          {
               result = atof(numbers);
               lcd.print(result);
               Serial.println(result);
```

```
            index = 0;
            numbers[index] = '\0';
        }
        else if(key == 'A')
        {
            index = 0;
            confirm = 1;
            numbers[index] = '\0';
            return result;
        }
        else if(key == 'B')
        {
            confirm2 = 1;
        }
    }
}

void GetStarted()
{
    char key = kpd.getKey();
    if(key != NO_KEY)
    {
        if(key == 'A')
        {
            confirm2 = 1;
        }
    }
}
```

BIODATA PENULIS

Ahmad Fauzi Aulia adalah anak kedua dari dua bersaudara. Meski lahir di Sleman, Yogyakarta pada tanggal 3 Oktober 1997, penulis tumbuh dan telah hampir 16 tahun hidup di Sidoarjo. Setelah lulus dari SDN Tropodo III, penulis melanjutkan pendidikan di SMP Negeri 1 Waru dan SMA Negeri 1 Waru. Pada tahun 2015, penulis diterima di Departemen Teknik Elektro ITS Surabaya melalui jalur SBMPTN. Selama masa perkuliahan, penulis banyak berkutat di bidang organisasi seperti menjadi Staff Piltek BEM FTI ITS, Staff VSA Himatektro ITS, turut serta dalam kepengurusan CnO dan SC Himatektro ITS, serta menjadi salah satu Tim Penyusun Materi Mubes V. Penulis punya mimpi besar untuk menjadi seorang penulis novel terkenal. Meskipun berkuliah di Teknik Elektro, hal itu tidak menyurutkan mimpi besar penulis untuk tetap berkarya dan berkembang dimanapun dan kapanpun.

Email : ahmadfauzi9d01@gmail.com
Hp/WA : 0816774588