TUGAS AKHIR – EE 184801

VISION SYSTEM UNTUK IDENTIFIKASI *SHAGAI* PADA ABU ROBOCON 2019

Fajar Luhung Parasdyo
NRP 07111540000045

Dosen Pembimbing
Dr. Ir. Djoko Purwanto, M.Eng.
Fajar Budiman, ST., M.Sc.

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

TUGAS AKHIR – EE 184801

VISION SYSTEM UNTUK IDENTIFIKASI *SHAGAI* PADA ABU ROBOCON 2019

Fajar Luhung Parasdyo
NRP 07111540000045

Dosen Pembimbing
Dr. Ir. Djoko Purwanto, M.Eng.
Fajar Budiman, ST., M.Sc.

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

FINAL PROJECT – EE 184801

VISION SYSTEM FOR SHAGAI IDENTIFICATION IN ABU ROBOCON 2019

Fajar Luhung Parasdyo
NRP 07111540000045

Supervisor
Dr. Ir. Djoko Purwanto, M.Eng.
Fajar Budiman, ST., M.Sc.

ELECTRICAL ENGINEERING DEPARTMENT
Faculty of Electrical Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2019

PERNYATAAN KEASLIAN TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul "***Vision System* untuk Identifikasi *Shagai* pada ABU Robocon 2019**" 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, 11 Juli 2019

Fajar Luhung Parasdyo
NRP 07111540000045

-----Halaman ini sengaja dikosongkan----

LEMBAR PENGESAHAN
VISION SYSTEM UNTUK IDENTIFIKASI *SHAGAI* PADA ABU ROBOCON 2019

TUGAS AKHIR

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

Menyetujui:

Dosen Pembimbing I	Dosen Pembimbing II
Dr. Ir. Djoko Purwanto, M.Eng.	Fajar Budiman, ST., M.Sc.
NIP. 198103252010121002	NIP. 198607072014041001

SURABAYA
JULI, 2019

-----Halaman ini sengaja dikosongkan-----

Vision System untuk Identifikasi *Shagai* pada ABU Robocon 2019

Nama : Fajar Luhung Parasdyo
Pembimbing : 1. Dr. Ir. Djoko Purwanto, M.Eng.
2. Fajar Budiman, ST., M.Sc.

ABSTRAK

ABU Robocon (Asia-Pacific Broadcasting Union Robot Contest) merupakan kontes robot tingkat Asia-Pasifik yang diselenggarakan oleh Asia-Pacific Broadcasting Union (ABU). Pada tahun 2019, ABU Robocon diselenggarakan di Mongolia dan mengangkat tema "*Great Urtuu*". Pada tema ini, ada dua robot yang dipakai yaitu robot manual dan robot otomatis. Inti permainannya adalah robot manual bertugas membawa objek bernama "Gerege" melewati halang rintang, lalu memberikannya pada robot otomatis. Setelah itu, robot otomatis berjalan melewati halang rintang hingga sampai di suatu area bernama "Mountain Urtuu", di situ robot otomatis harus menunggu robot manual untuk melempar objek bernama "*Shagai*". Setiap lemparan akan menghasilkan poin sesuai dengan warna permukaan atas pada *Shagai* saat mendarat. Bernilai 50 poin untuk warna emas, 40 poin untuk perak, dan 20 poin untuk biru atau merah (sesuai warna tim). Setiap tim harus mendapatkan minimal 50 poin agar robot otomatis boleh melaju hingga *finish*. Pada tugas akhir ini dibuat *vision system* untuk mengidentifikasi *Shagai* menggunakan kamera. Sistem ini dirancang dalam dua bagian, yaitu deteksi *Shagai* beserta posisinya pada citra dan identifikasi warna permukaan atas *Shagai* menggunakan segmentasi warna. Deteksi *Shagai* beserta posisinya pada citra dirancang menggunakan metode *Deep Learning Object Detection*. Sedangkan sistem identifikasi warna permukaan atas *Shagai* menggunakan metode segmentasi warna dalam pengolahan citra digital. Dari hasil pengujian, sistem untuk mengidentifikasi *Shagai* memiliki total akurasi sebesar 91,15% untuk *Shagai* merah dan 90,8% untuk *Shagai* biru.

Kata Kunci: *Vision System, Object Detection, Shagai,* ABU Robocon 2019

-----Halaman ini sengaja dikosongkan-----

Vision System for Shagai Identification in ABU Robocon 2019

Student's name : Fajar Luhung Parasdyo
Supervisor : 1. Dr. Ir. Djoko Purwanto, M.Eng.
2. Fajar Budiman, ST., M.Sc.

ABSTRACT

ABU Robocon (Asia Pacific Broadcasting Union Robot Contest) is an Asia-Pacific level robot contest organized by the Asia-Pacific Broadcasting Union (ABU). In 2019, ABU Robocon was held in Mongolia and raised the theme "Great Urtuu". In this theme, there are two robots used, namely manual robots and automatic robots. The core of the game is a manual robot tasked with carrying an object called "Gerege" past the obstacle, then giving it to an automatic robot. After that, the robot automatically walks through the obstacle until it reaches an area called "Mountain Urtuu", where the automatic robot must wait for the manual robot to throw an object named "Shagai". Each throw will produce points according to the color of the top surface on Shagai when landing. Worth 50 points for gold, 40 points for silver, and 20 points for blue or red (according to team colors). Each team must get a minimum of 50 points so that the automatic robot can advance to the finish. In this final project, a vision system was created to identify Shagai using a camera. This system is designed in two parts, the detection of Shagai and its position on the image, and identification of the upper surface color of Shagai using color segmentation. Detection of Shagai and its position in the image is designed using the Deep Learning Object Detection method. Whereas the surface color identification system above Shagai uses the color segmentation method in digital image processing. From the test results, the system for identifying Shagai has a total accuracy of 91.15% for Shagai with red color and 90.8% for Shagai with blue color.

Keywords: *Vision System, Object Detection, Shagai,* ABU Robocon 2019

-----Halaman ini sengaja dikosongkan-----

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Tuhan Yang Maha Esa karena berkat rahmat dan karunia-Nya penulis dapat menyelesaikan tugas akhir yang berjudul "***Vision System* untuk Identifikasi *Shagai* pada ABU Robocon 2019**" Pada kesempatan ini penulis menyampaikan terima kasih atas segala bantuan dan dukungannya yang telah diberikan selama proses pembuatan tugas akhir ini kepada:

1. Kedua orang tua tercinta, Bapak Sukono dan Ibu Bangun Sri Utami yang selalu memberikan do'a dan dukungan yang sangat berarti dalam keadaan apapun. Semoga Allah selalu memberikan kesehatan.
2. Bapak Dr. Ir. Djoko Purwanto, M.Eng., PhD. dan Bapak Fajar Budiman, ST., M.Sc. selaku Dosen Pembimbing atas segala bimbingan ilmu, moral, dan spiritual dari awal hingga terselesaikannya tugas akhir,
3. Seluruh Staff/Karyawan/Dosen Departemen Teknik Elektro yang telah memberikan banyak ilmu dan menciptakan suasana belajar yang mendukung.
4. Teman-teman e55 yang telah berjuang bersama semenjak awal masuk perkuliahan.
5. Semua pihak yang telah membantu penulis dalam menyelesaikan
   tugas akhir ini yang tidak dapat disebutkan satu per satu.

Penulis berharap semoga tugas akhir ini dapat bermanfaat dalam pengembangan keilmuan di kemudian hari.

Surabaya, 11 Juli 2019

Penulis

-----Halaman ini sengaja dikosongkan-----

DAFTAR ISI

-----Halaman ini sengaja dikosongkan-----

DAFTAR GAMBAR

DAFTAR TABEL

-----Halaman ini sengaja dikosongkan-----

BAB I
PENDAHULUAN

Pada bab ini akan dibahas mengenai latar belakang dari penelitian tugas akhir. Berdasarkan latar belakang tersebut dapat dirumuskan permasalahan dan tujuan tugas akhir. Selanjutnya dibahas mengenai metodologi, sistematika, dan relevansi tugas akhir yang dikerjakan.

1.1 Latar Belakang

Pada ABU Robocon 2019, terdapat dua robot yang terdiri dari robot manual dan robot otomatis. Kedua robot tersebut memiliki sekumpulan tugas pada pertandingan, salah satunya adalah mengidentifikasi *Shagai* (sebuah benda yang digunakan dalam pertandingan) untuk mengetahui warna permukaan atasnya. Hal ini dapat dilakukan dengan cara melihat langsung menggunakan mata kita, lalu operator mengirimkan perintah ke robot manual, setelah itu robot manual mengirimkan suatu tanda ke robot otomatis untuk penentuan keputusan selanjutnya. Cara ini membutuhkan mekanisme tambahan pada robot manual untuk memberikan tanda ke robot otomatis. Cara ini juga kurang efektif karena harus mengirimkan perintah melalui robot manual terlebih dahulu. Waktu yang terbuang dapat menjadi penyebab kekalahan pada pertandingan ini. Oleh karena itu, dicari metode lain yang lebih efektif untuk menghemat waktu. Pada tugas akhir ini akan dibuat sistem berbasis kamera untuk identifikasi *Shagai*.

Object Detection merupakan sebuah metode untuk mendeteksi dan mengetahui posisi objek pada sebuah citra. Metode ini digunakan untuk mendeteksi *Shagai* milik tim kita. Setelah *Shagai* dapat dideteksi, digunakanlah pengolahan citra digital untuk mengetahui warna permukaan atasnya. Hal ini dilakukan untuk menentukan langkah selanjutnya yang harus dilakukan robot otomatis. Metode *Object Detection* digunakan karena cepat dan memiliki tingkat akurasi yang tinggi, sehingga selain dapat menghemat waktu pada pertandingan, juga menciptakan sistem yang *robust* pada robot otomatis ini.

1.2 Permasalahan

Permasalahan dalam tugas akhir ini adalah:

1. Bagaimana cara untuk mendeteksi *Shagai* menggunakan kamera.

2. Bagaimana menentukan warna permukaan atas *Shagai* menggunakan kamera.

1.3 Tujuan

Tujuan dalam tugas akhir ini adalah:

1. Mendapatkan hasil deteksi *Shagai* menggunakan kamera.
2. Mengetahui warna permukaan atas *Shagai* berdasarkan hasil deteksi dan pengolahan citra.

1.4 Batasan Masalah

Batasan masalah dalam tugas akhir ini adalah:

1. Lingkungan disesuaikan dengan arena ABU Robocon 2019.
2. Warna *Shagai* disesuaikan dengan aturan ABU Robocon 2019.
3. *Shagai* tidak terhalang objek lain.

1.5 Metodologi

Metodologi yang digunakan dalam menyusun penelitian tugas akhir ini adalah sebagai berikut:

1. Studi Literatur

Studi literatur berisi tentang pengumpulan beserta pengkajian teori, data dan metode yang relevan terhadap tugas akhir ini. Literatur yang digunakan memiliki batasan tertentu yaitu bersumber dari paper, jurnal, buku ataupun artikel yang terpercaya. Literatur ini bertujuan untuk memperkuat landasan teori dan sebagai dasar untuk mengerjakan tugas akhir ini.

2. Perancangan Sistem

Perancangan sistem dilakukan agar sistem dapat mengidentifikasi *Shagai*. Awalnya, sistem akan melakukan proses *learning* dari *dataset* gambar *Shagai*. Setelah proses tersebut selesai, sistem akan diuji menggunakan gambar lain yang belum pernah digunakan dalam proses *learning* untuk mengetahui seberapa besar akurasinya.

3. Pengambilan *Dataset*

Pengambilan *dataset* dilakukan untuk mendapatkan kumpulan gambar *Shagai* dalam berbagai sudut dan posisi. Nantinya kumpulan gambar ini akan digunakan untuk melakukan proses pelatihan.

4. Proses Pelatihan

Pada tahap ini dilakukan proses pelatihan untuk mengajari/melatih sistem yang akan dibuat agar bisa melakukan identifikasi *Shagai*. Nantinya akan didapat sebuah model yang dapat melakukan prediksi terhadap masukan baru. Dalam kasus tugas akhir ini masukan baru yang

dimaksud adalah gambar baru dari video yang diambil kamera saat kompetisi.

5. Pengujian Sistem

Pada tahap ini dilakukan pengujian keseluruhan sistem untuk mendapatkan data-data yang diperlukan untuk bahan analisa dan evaluasi sistem nantinya. Yang diperlukan adalah data akurasi dalam mendeteksi *Shagai* dan juga data akurasi dalam mengidentifikasi warna.

6. Analisa dan Evaluasi Sistem

Pada tahap ini, akan dilakukan analisa terhadap data yang didapatkan sehingga tahu seberapa bagus sistem yang telah dibuat. Analisa dilakukan pada data akurasi dalam mendeteksi *Shagai* dan identifikasi warnanya. Lalu dilakukan evaluasi serta revisi apabila hasil sistem kurang memuaskan.

7. Penyusunan Laporan

Penyusunan laporan akan dilakukan seiring dengan tahap-tahap lainnya. Isinya berkaitan dengan tugas akhir yang dikerjakan, meliputi pendahuluan, studi literatur, perancangan dan pembuatan sistem, pengujian dan analisa serta penutup.

1.6 Sistematika Penulisan

Penulis membagi laporan penelitian ini menjadi lima bab yang terhubung satu sama lain. Hal ini untuk menghindari kesalahan interpretasi terhadap isi di dalam laporan. Penjelasan tentang masing-masing bab dibuat dengan sistematika penulisan sebagai berikut:

BAB 1 PENDAHULUAN

Pada bab ini dijelaskan tentang latar belakang, permasalahan, batasan masalah, tujuan, metodologi, sistematika penulisan, serta relevansi dari penelitian yang dilakukan.

BAB 2 TINJAUAN PUSTAKA

Pada bab ini dibahas mengenai tinjauan pustaka yang membantu penelitian. Teori yang dapat membantu penelitian ini antara lain adalah jaringan saraf tiruan, *object detection*, dan pengolahan citra digital.

BAB 3 PERANCANGAN SISTEM

Pada bab ini membahas tentang perancangan sistem yang akan digunakan pada tugas akhir ini. Perancangan sistem ini meliputi perancangan mekanik dan program.

BAB 4 PENGUJIAN DAN ANALISIS

Pada bab ini dibahas hasil deteksi *Shagai* dan juga identifikasi warna *Shagai*. Selanjutnya dilakukan analisa berdasarkan data hasil pengujian.

BAB 5 PENUTUP

Bab ini berisi kesimpulan dan saran dari hasil pembahasan yang telah diperoleh.

1.7 Relevansi

Penelitian diharapkan dapat memberikan manfaat bagi Tim KRAI Robotika ITS dalam perlombaan ABU Robocon 2019 dengan menggunakan *vision system* ini untuk mempermudah identifikasi *Shagai* pada robot otomatis.

BAB II
TINJAUAN PUSTAKA

Bab ini membahas tentang teori penunjang yang dapat membantu penelitian. Teori ini nantinya akan digunakan sebagai dasar untuk perancangan sistem pada bab III.

2.1 ABU Robocon 2019

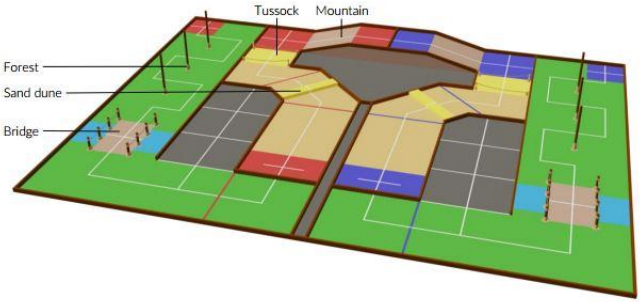
ABU Robocon (Asia-Pacific Broadcasting Union Robot Contest) merupakan kontes robot tingkat Asia-Pasifik. Kontes ini diadakan setiap satu tahun sekali dengan tuan rumah yang berbeda-beda setiap tahunnya. Tema dari kontes ini juga selalu berbeda-beda setiap tahunnya mengikuti negara yang menjadi tuan rumah. Pada tahun 2019 ini, ABU Robocon diselenggarakan di Ulaanbaatar, Mongolia dan mengangkat tema "Great Urtuu".

Tema permainan dari ABU Robocon 2019 terinspirasi dari "Urtuu", sistem *messenger relay* yang digunakan untuk memberikan informasi secara cepat yang pertama kali diinovasi oleh orang nomaden Mongolia [1]. Untuk bertukar informasi dalam jarak jauh, orang Mongolia telah menggunakan sistem Urtuu sebagai pembawa pesan untuk istirahat (memberi makan, mengganti kuda, dll.), dan dalam beberapa kasus, meneruskan ke pembawa pesan lain. Dengan menggunakan sistem Urtuu, seorang kurir dapat melakukan perjalanan dalam jarak 400 kilometer per hari [1]. Saat ini, kita sedang mengalami perkembangan pertukaran dan berbagi pengetahuan dan informasi secara masif dan tiba-tiba. Sistem Urtuu ini adalah penemuan penting yang membuka pintu baru bagi kami untuk bertukar dan berbagi pengetahuan tanpa memandang ruang. Berdasarkan konsep ini, ABU Robocon 2019 Ulaanbaatar dirancang untuk mempromosikan gagasan "Sharing the knowledge" [1].

Pertandingan antara tim Merah dan Biru paling lama berlangsung tiga menit. Setiap tim memiliki satu robot manual yang dikenal sebagai *Messenger-Robot 1*, dan satu robot otomatis yang dikenal sebagai *Messenger-Robot 2* [1]. Robot otomatis memiliki empat kaki seperti kuda dan tidak diperbolehkan menggunakan roda. Robot manual membawa sebuah objek yang bernama Gerege. Robot manual berjalan di sepanjang *Forest*, *Bridge*, dan melintasi Jalur 1 di sebelah *Gobi urtuu*, yang merupakan titik awal dari robot otomatis. Setelah *Messenger-Robot 1* mencapai *Gobi urtuu*, *Messenger-Robot 1* memberikan *Gerege* ke *Messenger-Robot 2* di *Gobi urtuu*. Setelah *Messenger-Robot 2* berhasil menerima *Gerege*, robot otomatis dapat berjalan di *Gobi Area*.

Messenger-Robot 2 harus berjalan dengan empat kaki, seperti kuda, dan tidak boleh menggunakan roda untuk bergerak. *Messenger-Robot 2* melewati *Sand Dune* dan *Tussock*, lalu ke *Mountain urtuu*. Setelah *Messenger-Robot 2* mencapai *Mountain urtuu*, *Messenger-Robot 1* dapat memasuki *Throwing zone* untuk melempar *Shagai*, dan harus mendapatkan 50 poin atau lebih. Jika *Messenger-Robot 1* menghasilkan 50 poin atau lebih, *Messenger-Robot 2* diperbolehkan untuk mendaki Gunung. Setelah itu, jika mencapai zona *Uukhai* dan mengangkat *Gerege* terlebih dahulu, tim tersebut adalah pemenangnya, yang disebut "UUKHAI".

Gambar 2.1 Lapangan permainan - area dan zona **[1]**

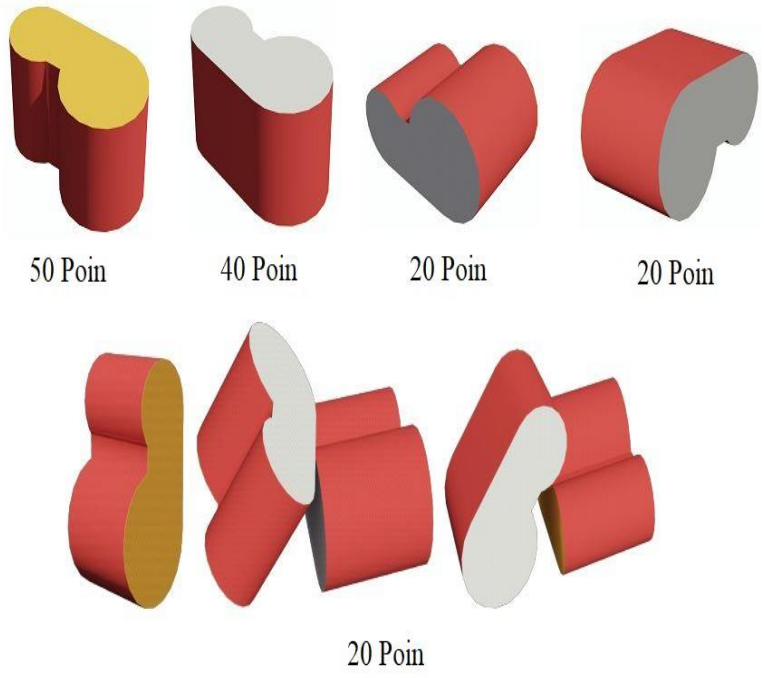


Gambar 2.2 Lapangan permainan – objek **[1]**

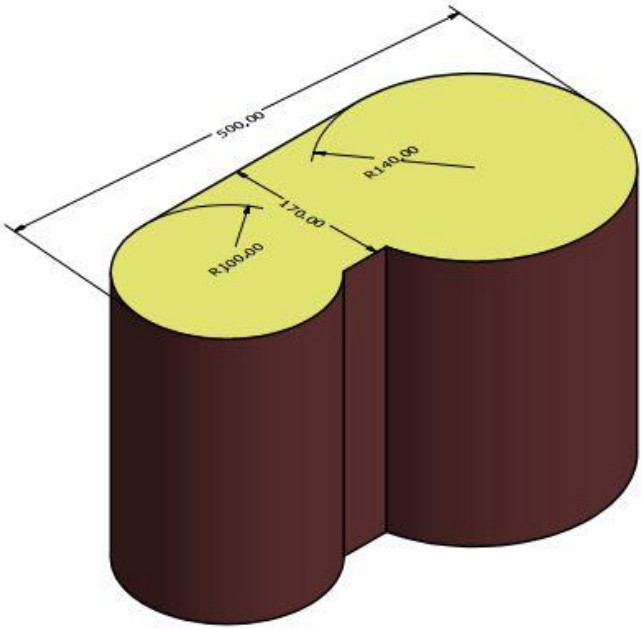
2.2 *Shagai*

Shagai adalah sebuah objek yang dipakai dalam perlombaan ABU Robocon 2019. *Shagai* memiliki bentuk yang unik dan memiliki warna yang berbeda pada tiap sisinya. Nantinya, warna tersebutlah yang akan menentukan poin pada saat pertandingan. Poin ditentukan berdasarkan warna sisi atas *Shagai* saat mendarat pada arena pertandingan. Detail untuk penentuan poin dari *Shagai* dapat dilihat pada **gambar 2.3**.

Pada ABU Robocon 2019, *Shagai* dibedakan menjadi dua, yaitu *Shagai* dengan selimut berwarna merah dan *Shagai* dengan selimut berwarna biru. Warna tersebut disesuaikan dengan warna tim yang bertanding, yaitu tim merah dan tim biru. Detail bentuk dan dimensi *Shagai* dapat dilihat pada **gambar 2.4**.



Gambar 2.3 Penentuan Poin *Shagai* **[2]**



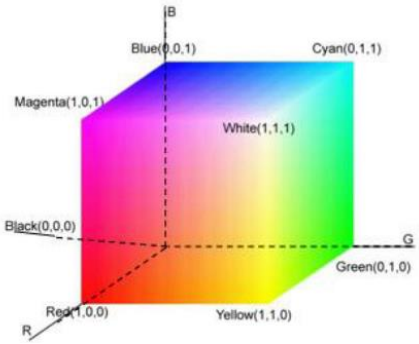
Gambar 2.4 Bentuk dan Dimensi *Shagai* **[3]**

2.3. Pengolahan Citra Digital

Pengolahan citra digital masih merupakan sebuah kebutuhan yang besar dalam bidang penelitian. Penelitian yang terkait dengan pengolahan citra digital dapat berupa komponen warna, tekstur dan pola. Pengolahan citra digital bertujuan untuk memperoleh informasi atau deskripsi yang terdapat pada objek [4]. Berikut ini merupakan beberapa hal dasar yang perlu diketahui dalam pengolahan citra digital:

2.3.1 Ruang Warna RGB

Ruang warna RGB adalah ruang warna yang paling banyak digunakan, biasanya pada komputer dan kamera digital. Ruang warna ini didasarkan pada campuran dari tiga warna primer R, G dan B yang dinyatakan secara fisik [5].

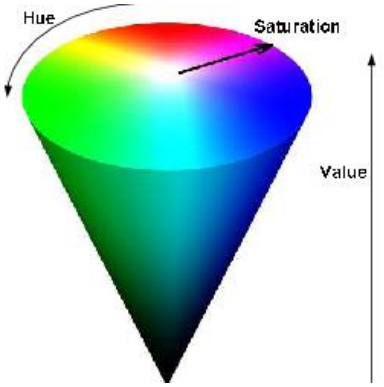


Gambar 2.5 Kubus RGB **[6]**

Ruang warna RGB dapat dimodelkan dalam bentuk kubus tiga dimensi. Semua warna yang bisa didapatkan dari kombinasi warna R, G, dan B telah direpresentasikan pada kubus RGB seperti pada **gambar 2.5**. Warna hitam dalam RGB direpresentasikan dalam nilai (0, 0, 0) untuk nilai R, G, dan B secara urut. Warna putih direpresentasikan dalam nilai maksimal yaitu (255, 255, 255) untuk nilai R, G, dan B secara urut, apabila menggunakan resolusi 8 bit. Namun hasil kombinasi ini tidak disukai karena tidak sesuai dengan warna aslinya. Sehingga penelitian terkait gambar digital mengusulkan ruang warna HSV yang merupakan representasi dari *Hue, Saturation,* dan *Value* [4].

2.3.2 Ruang Warna HSV

Hasil kombinasi warna pada ruang warna RGB memiliki sebuah masalah, yaitu warna yang terbentuk dari hasil kombinasi tidak sepenuhnya merepresentasikan warna aslinya, untuk mengatasi hal itu maka diterapkanlah transformasi non-linier pada ruang warna RGB, salah satu hasilnya adalah ruang warna HSV. Ruang warna HSV dinyatakan oleh tiga elemen yaitu, *hue*, *saturation*, dan *value*. Ruang warna HSV direpresentasikan dalam sebuah model kerucut terbalik. *Hue* merupakan tipe warna, seperti: merah, kuning [5]. Tipe warna ini ditentukan oleh nilai dari 0 sampai 360 derajat pada kerucut warna HSV seperti pada **gambar 2.6.** *Saturation* merupakan kemurnian warna, dimana dalam kerucut HSV ditentukan oleh nilai dari 0 sampai 100% [5]. *Value* merupakan tingkat kecerahan, dimana dalam kerucut HSV ditentukan oleh nilai dari 0 sampai 100% [5].



Gambar 2.6 Kerucut warna HSV **[6]**

Ruang warna ini termasuk dalam sistem warna non-linier. Metode representasi warna HSV sesuai dengan persepsi manusia tentang warna. Setiap elemen ruang warna terpisah, sehingga cocok untuk pengolahan citra digital [7].

2.3.3 Citra Biner

Citra biner merupakan citra yang pikselnya hanya memiliki dua nilai, yaitu minimal dan maksimal. Kedua nilai tersebut merupakan representasi warna hitam dan warna putih. Warna hitam diwakili oleh nilai minimal, yaitu nilai 0. Warna putih diwakili oleh nilai maksimal, yaitu 255 (misalkan menggunakan resolusi 8 bit). Dalam pengolahan citra digital, citra biner biasanya digunakan untuk memisahkan antara objek dan latar belakang.

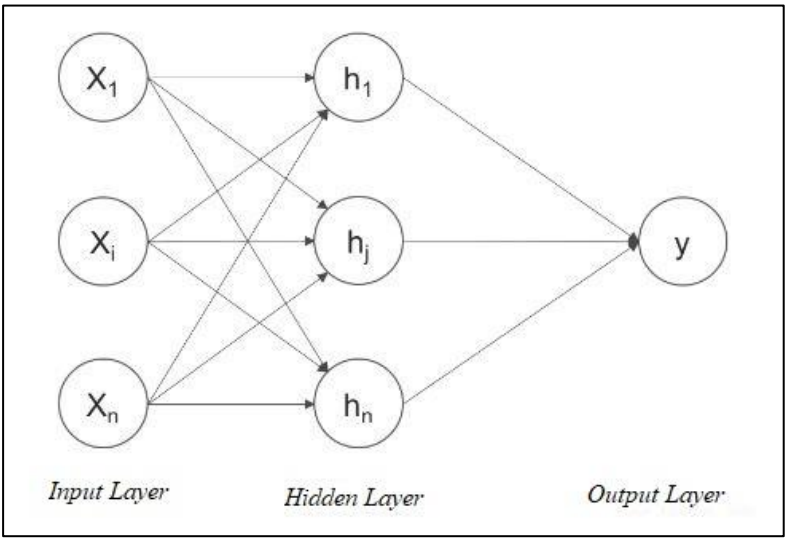
2.4 Jaringan Saraf Tiruan

Pembelajaran mesin (*machine learning*) adalah sebuah penerapan metode yang membantu perangkat lunak melakukan tugas tanpa pemrograman atau aturan yang eksplisit. Dengan pemrograman komputer tradisional, seorang programmer menentukan aturan yang harus digunakan komputer. Namun, pembelajaran mesin menerapkan pola pikir yang berbeda. Pembelajaran mesin jauh lebih berfokus pada analisis data daripada pemrograman. Pemrogram menyediakan serangkaian contoh dan komputer mempelajari pola dari data. Pembelajaran mesin dapat dianggap sebagai “pemrograman dengan data” [8].

Jaringan saraf tiruan (Artificial Neural Network) merupakan salah satu metode pembelajaran mesin yang paling banyak digunakan saat ini.

Jaringan saraf tiruan terinspirasi dari cara kerja neuron pada otak manusia. Neuron pada otak manusia melakukan komunikasi satu sama lain untuk mengirim dan menerima sebuah sinyal informasi. Dalam penerapannya, jaringan saraf tiruan tersusun dari beberapa lapisan (*layer*), yaitu *input layer*, *output layer*, dan minimal satu *hidden layer*. Masing-masing lapisan memiliki *node* yang disebut dengan neuron. Masing-masing lapisan juga memiliki nilai bobot yang disebut *weight*, dan suatu nilai konstanta yang disebut *bias*. Struktur dari jaringan saraf tiruan dapat dilihat pada **gambar 2.7**.

Jaringan saraf tiruan perlu dilatih untuk mendapatkan model yang diinginkan. Proses pelatihan ini sebenarnya adalah proses dalam mencari nilai dari *weight* dan *bias* yang tepat pada tiap layernya. Dengan nilai weight dan bias yang tepat, maka akan didapatkan nilai keluaran yang sesuai untuk setiap nilai masukan yang diberikan.

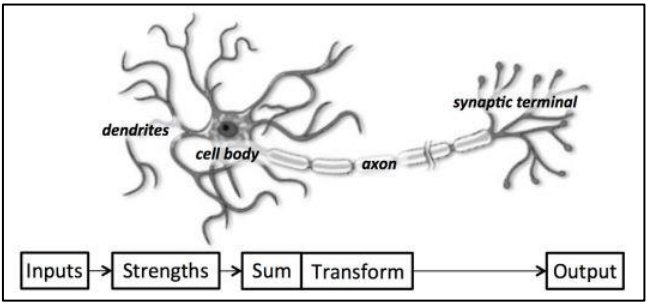


Gambar 2.7 Struktur jaringan saraf tiruan

2.4.1 Neuron

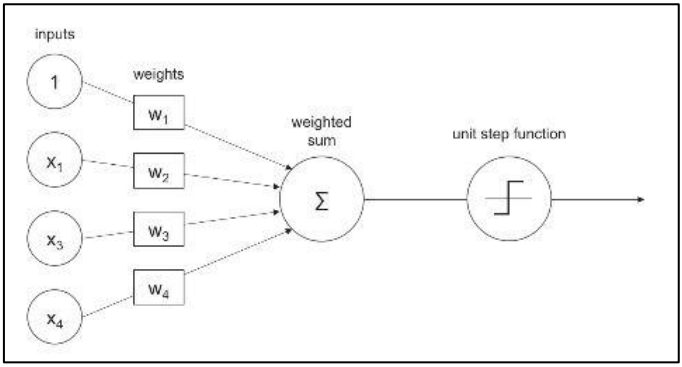
Unit dasar otak manusia adalah neuron. Neuron digunakan untuk menerima informasi dari neuron lain, memproses informasi ini dengan cara yang unik, dan mengirimkan hasilnya ke sel lain. Neuron menerima sinyal masukan melalui dendrit. Lalu sinyal masukan ini diperkuat atau

dilemahkan (pembobotan) pada setiap koneksi secara dinamis berdasarkan seberapa sering digunakan. Setelah diberi bobot oleh kekuatan koneksi masing-masing, sinyal masukan dijumlahkan bersama dalam tubuh sel. Jumlah ini kemudian diubah menjadi sinyal baru yang disebarkan sepanjang akson sel dan dikirim ke neuron lain [9]. Dari konsep neuron pada otak manusia inilah, jaringan saraf tiruan dibuat. Struktur neuron otak manusia dapat dilihat pada **gambar 2.8**.



Gambar 2.8 Struktur Neuron pada Otak Manusia **[9]**

Neuron atau perceptron pada jaringan saraf tiruan meniru apa yang dilakukan neuron pada otak manusia. Neuron pada jaringan saraf tiruan mengambil beberapa masukan dan melakukan penjumlahan berbobot menggunakan *weight* untuk menghasilkan keluaran [10]. *Weight* pada neuron ditentukan selama proses pelatihan dan didasarkan pada data pelatihan. Struktur neuron pada jaringan saraf tiruan ditunjukkan pada **gambar 2.9**.

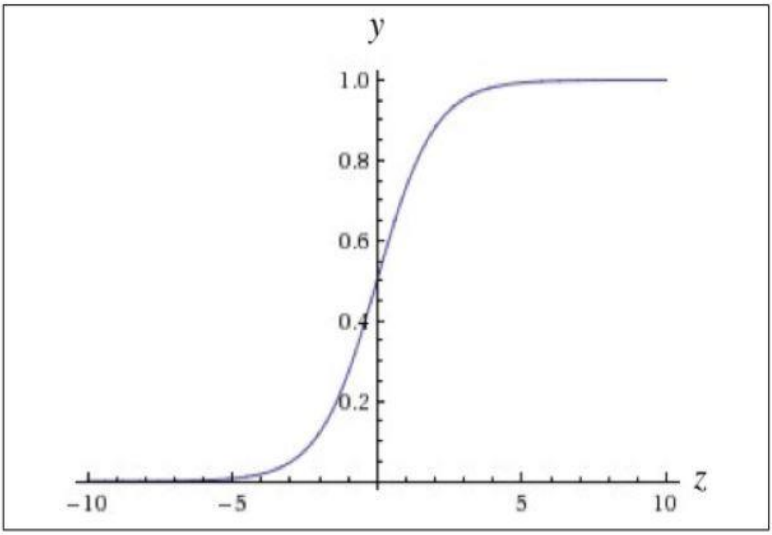


Gambar 2.9 Struktur Neuron pada Jaringan Saraf Tiruan

2.4.2 Fungsi Aktivasi

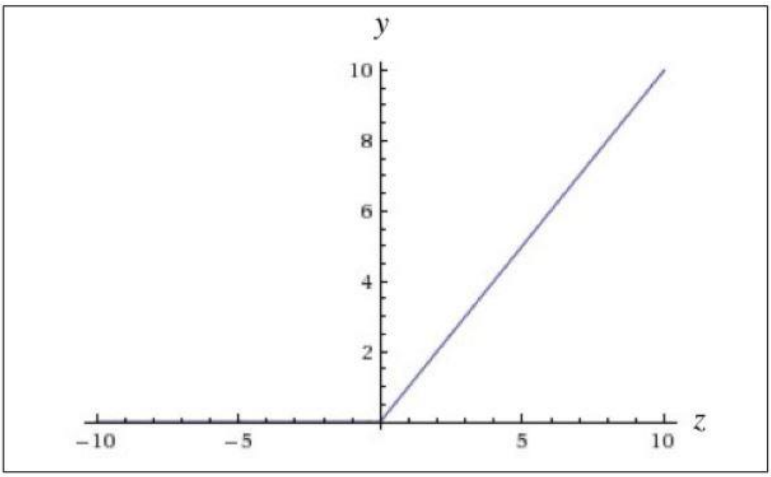
Fungsi aktivasi membuat jaringan saraf menjadi nonlinier. Fungsi aktivasi memutuskan apakah perceptron harus diaktifkan atau tidak [10]. Selama aktivasi pelatihan, fungsi memainkan peran penting dalam menyesuaikan gradien [10]. Fungsi aktivasi seperti sigmoid, ditunjukkan pada **gambar 2.10**, melemahkan nilai dengan magnitudo lebih tinggi. Perilaku nonlinear dari fungsi aktivasi ini memungkinkan model mempelajari fungsi yang kompleks. Sebagian besar fungsi aktivasi adalah fungsi kontinu dan dapat diturunkan, kecuali fungsi ReLU (Rectified Linear Unit) yang ditunjukkan pada **gambar 2.11**. Fungsi ReLU tidak dapat diturunkan pada nilai 0. Fungsi kontinu memiliki perubahan kecil pada output untuk setiap perubahan kecil pada input. Fungsi diferensial memiliki turunan yang ada di setiap titik dalam domain. Terdapat beberapa fungsi aktivasi yang biasanya digunakan pada jaringan saraf tiruan seperti, sigmoid, tanh, dan ReLU.

Sigmoid dapat dianggap sebagai fungsi step yang diperhalus dan oleh karena itu dapat diturunkan. Sigmoid berguna untuk mengonversi nilai apa pun menjadi probabilitas dan dapat digunakan untuk klasifikasi biner [10]. Sigmoid menerima sinyal masukan dalam kisaran nilai 0 hingga 1, seperti yang ditunjukkan pada **gambar 2.10**.



Gambar 2.10 Grafik Fungsi Sigmoid **[10]**

ReLu dapat membiarkan angka besar lewat. ReLU memetakan masukan x ke max(0, x), yaitu memetakan masukan negatif ke 0, dan masukan positif adalah keluaran tanpa perubahan seperti yang ditunjukkan pada **gambar 2.10**.



Gambar 2.11 Grafik Fungsi ReLU

2.4.3 *Feed Forward*

Feed Forward adalah langkah untuk menyalurkan atau merambatkan maju masukan pada jaringan saraf tiruan. Untuk mempercepat proses pelatihan, nilai masukan perlu dinormalisasi terlebih dahulu [6].

$$\boldsymbol{a}^0 = \boldsymbol{p} \tag{2.1}$$

$$\boldsymbol{a}^{m+1} = \boldsymbol{f}^{m+1}(\boldsymbol{w}^{m+1}\boldsymbol{a}^m + \boldsymbol{b}^{m+1}), \quad m = 0,1,\cdots,M-1 \tag{2.2}$$

$$\boldsymbol{a} = \boldsymbol{a}^M \tag{2.3}$$

Dimana:
$\boldsymbol{a}^0$: masukan jaringan syaraf tiruan
$\boldsymbol{a}^M$: keluaran jaringan syaraf tiruan

$\boldsymbol{a}^i$: keluaran *node layer* ke-i
$\boldsymbol{p}$: masukan yang sudah dinormalisasi
$\boldsymbol{f}^i$: fungsi aktivasi *layer* ke-i
$\boldsymbol{w}^i$: *weight layer* ke-i
$\boldsymbol{b}^i$: *bias layer* ke-i

2.4.4 *Backpropagation*

Algoritma *backpropagation* umumnya digunakan untuk pelatihan jaringan saraf tiruan [6]. *Weight* dan *bias* diperbarui dari belakang dimulai dari *error* yang terhitung pada *output layer*, lalu berlanjut ke *hidden layer* paling belakang, hingga selesai pada *hidden layer* paling depan.

$$\boldsymbol{s}^M = -2\dot{\boldsymbol{F}}^M(\boldsymbol{n}^M)(\boldsymbol{t}-\boldsymbol{a}) \tag{2.4}$$

$$\boldsymbol{s}^m = \dot{\boldsymbol{F}}^m(\boldsymbol{n}^m)(\boldsymbol{w}^{m+1})^T\boldsymbol{s}^{m+1}, \quad m = M-1,\cdots,2,1 \tag{2.5}$$

$$\dot{\boldsymbol{F}}^m(\boldsymbol{n}^m) = \begin{bmatrix} \dot{f}^m(n_1^m) & 0 & \cdots & 0 \\ 0 & \dot{f}^m(n_2^m) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \dot{f}^m\left(n_{s^m}^m\right) \end{bmatrix} \tag{2.6}$$

$$\dot{f}^m\left(n_j^m\right) = \frac{\partial f^m(n_j^m)}{\partial n_j^m} \tag{2.7}$$

Dimana:
$\boldsymbol{a}$: keluaran jaringan syaraf tiruan
$\boldsymbol{t}$: keluaran data kalibrasi yang sudah dinormalisasi
$\boldsymbol{s}^M$: sensitifitas *layer* keluaran
$\boldsymbol{s}^i$: sensitifitas *layer* ke-i
$\boldsymbol{w}^i$: *weight layer* ke-i
$\boldsymbol{b}^i$: *bias layer* ke-i
$\dot{\boldsymbol{F}}^i$: turunan pertama fungsi aktivasi *layer* ke-i
$\boldsymbol{n}^i$: keluaran *node layer* ke-i

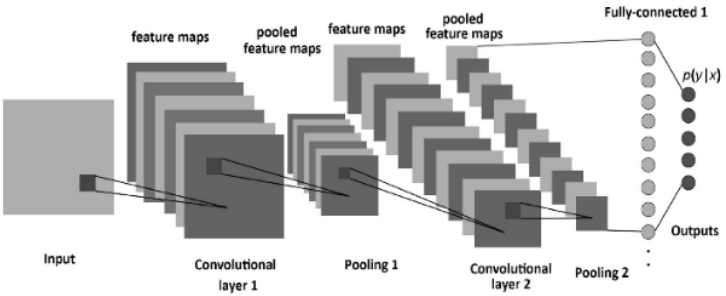
2.4.5 *Convolutional Neural Network*

Convolutional Neural Network (CNN) adalah jenis jaringan *feed-forward neural network* khusus yang mencakup lapisan *convolutional* dan *pooling* dalam arsitekturnya. Pola umum untuk arsitektur CNN adalah

memiliki *input layer*, beberapa kombinasi *convolutional layer* dan *pooling layer*, dan *fully-connected layer* yang terhubung dengan *output layer* [6]. Arsitektur jaringan CNN dapat dilihat pada **gambar 2.12**.

Konvolusi adalah konsep utama dibalik arsitektur CNN. Secara sederhana, konvolusi adalah operasi matematika yang menggabungkan informasi dari dua sumber untuk menghasilkan satu set informasi baru [6]. Secara khusus, ia menerapkan matriks khusus yang dikenal sebagai *kernel* ke *input*, untuk menghasilkan satu set matriks yang dikenal sebagai *feature maps*.

Pooling mengacu pada penghitungan statistik agregat atas wilayah dari *feature* yang dikonvolusi [6]. Dua jenis statistik agregat paling populer adalah maksimum dan rata-rata. Output dari penerapan *max-pooling* adalah maksimum wilayah yang dipilih, sedangkan output dari penerapan *average-pooling* adalah rata-rata dari angka-angka di wilayah tersebut.



Gambar 2.12 Arsitektur Jaringan CNN **[6]**

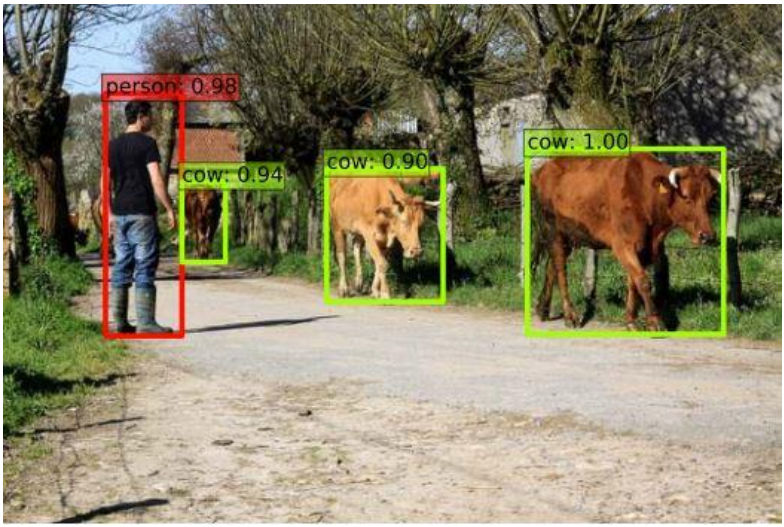
Arsitektur CNN telah terbukti sangat berhasil dalam memecahkan masalah yang melibatkan gambar, seperti *object detection* dan *object identification*.

2.5 *Object Detection*

Dengan kebangkitan kembali *neural network* pada tahun 2000-an, *deep learning* telah menjadi bidang penelitian yang sangat populer yang membuka jalan bagi pembelajaran mesin modern [9]. *Deep learning* adalah bagian dari bidang pembelajaran mesin, yang didasarkan pada gagasan belajar dari contoh. Sebagian besar model *deep learning* didasarkan pada jaringan saraf tiruan. Dalam pembelajaran mesin, alih-alih mengajar komputer sekumpulan aturan untuk menyelesaikan

masalah, pembelajaran mesin memberikan sebuah model yang dapat digunakan untuk mengevaluasi contoh, dan sekumpulan instruksi kecil untuk memodifikasi model ketika melakukan kesalahan [9]. Model *deep learning* dilatih dengan menggunakan data yang jumlahnya besar. Sekarang ini, *deep learning* telah banyak digunakan dalam aplikasi *computer vision* dan secara signifikan meningkatkan performanya. Contoh aplikasi *deep learning* pada *computer vision* adalah untuk klasifikasi citra dan deteksi objek.

Object Detection (deteksi objek) adalah metode untuk menemukan objek dalam sebuah citra dan menemukan posisinya yang direpresentasikan oleh *bounding box* (kotak pembatas) [11]. Contoh *object detection* dapat dilihat pada **gambar 2.3**. Metode ini banyak sekali digunakan dalam aplikasi *computer vision*, seperti untuk mendeteksi pejalan kaki, mendeteksi mobil, mendeteksi lampu lalu lintas, dan lain-lain.



Gambar 2.13 Contoh *Object Detection*

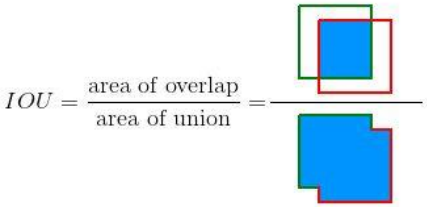
2.5.1 *Intersection Over Union* (IOU)

Intersection Over Union (IOU) merupakan sebuah *metric* (sistem pengukuran) untuk mengetahui seberapa bagus lokalisasi objek pada metode deteksi objek [10]. IOU merupakan luas irisan *bounding box* prediksi dengan *ground truth* dibagi luas gabungan *bounding box* prediksi

dengan *ground truth*. Rumus untuk menghitung IOU dapat dilihat pada **persamaan 2.8**. Adapun ilustrasi IOU dapat dilihat pada **gambar 2.14**, dimana *bounding box ground truth* berwarna hijau dan *bounding box* prediksi berwarna merah.

$$IOU = \frac{\text{luas}(B_p \cap B_{gt})}{\text{luas}(B_p \cup B_{gt})} \tag{2.8}$$

Dimana:
B_p : *bounding box* prediksi
B_{gt} : *bounding box ground truth*



Gambar 2.14 Ilustrasi IOU

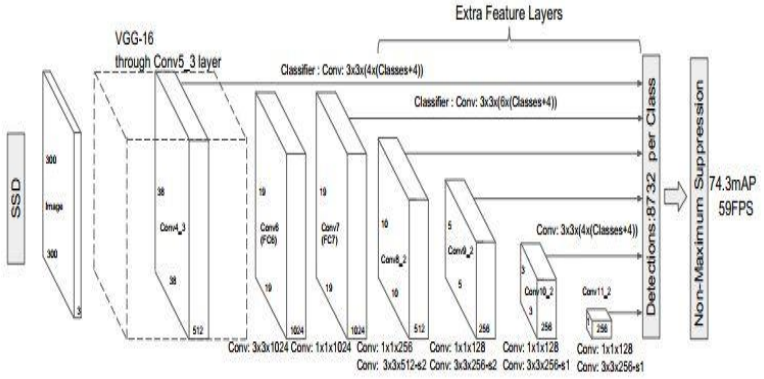
2.5.2 *Single Shot Detector*

Sistem deteksi objek yang canggih saat ini merupakan varian dari pendekatan berikut: memprediksi *bounding box*, sampel ulang piksel atau fitur untuk setiap *bounding box* dan menerapkan klasifikasi kualitas tinggi [12]. Meskipun akurat, pendekatan ini terlalu berat secara komputasi untuk sistem tertanam dan bahkan dengan perangkat keras kelas atas, serta terlalu lambat untuk aplikasi secara *real-time* [12]. Untuk itu *Single Shot Detector* (SSD) menawarkan jaringan yang lebih ringan dan cepat, tanpa kehilangan akurasi dalam melakukan deteksi objek. SSD mendiskritisasi bagian output dari *bounding box* menjadi satu set *default box* pada berbagai rasio dan skala per lokasi *feature map*. Pada saat prediksi, jaringan menghasilkan skor untuk keberadaan setiap kategori objek di setiap *default box* dan menghasilkan penyesuaian ke *box* untuk lebih cocok dengan bentuk objek. Peningkatan mendasar dalam kecepatan berasal dari menghilangkan *bounding box proposals* dan tahap sampel ulang piksel atau fitur.

Pendekatan SSD didasarkan pada *feed-forward convolutional network* yang menghasilkan beberapa *bounding box* dan skornya untuk setiap adanya kelas objek dalam *box* tersebut. Lapisan jaringan awal SSD

didasarkan pada arsitektur standar yang digunakan untuk klasifikasi gambar berkualitas tinggi (terpotong sebelum lapisan klasifikasi), yang disebut jaringan dasar. Setelah itu ditambahkan struktur tambahan ke jaringan untuk menghasilkan deteksi dengan fitur-fitur utama yaitu, *multi-scale feature maps* untuk ekstraksi fitur pada berbagai skala, *convolutional predictors* untuk menghasilkan prediksi deteksi setiap ditambahkan *feature layer*, serta *default boxes* dan *aspect ratio* untuk menghasilkan *default boxes* pada setiap *feature maps* yang memiliki ukuran berbeda-beda. Intinya, model SSD menambahkan beberapa lapisan fitur ke ujung jaringan dasar. Struktur jaringan SSD ditunjukkan pada **gambar 2.15**.

Single Shot Detector adalah objek detektor berdasarkan DNN (*Deep Neural Network*) tunggal. Detektor ini mencakup jaringan dasar VGG-16 yang terpotong sebelum *dense layers*, seperangkat lapisan konvolusional tambahan, lapisan *box generator* dan lapisan yang mengevaluasi output konvolusional dan menghasilkan deteksi objek yang diprediksi. Pada dasarnya, untuk setiap jaringan *forward pass*: 1) mengekstrak fitur pada skala yang berbeda; 2 untuk setiap skala *feature map*, menghasilkan satu set *box default*; 3) untuk setiap *box*, akan memprediksi empat *offset* dan empat varian (langkah lokalisasi); 4) untuk setiap prediksi *box*, akan memprediksi serangkaian skor kepercayaan (*confidence score*) satu untuk setiap kelas (langkah klasifikasi); 5) kemudian mengumpulkan prediksi untuk menghasilkan output deteksi objek [13].

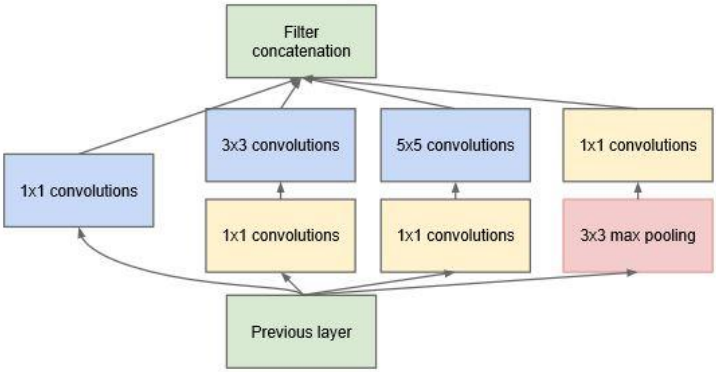


Gambar 2.15 Model SSD **[12]**

2.5.3 *Inception*

Inception merupakan salah satu arsitektur berbasis *deep convolutional neural network*. *Deep neural network* memiliki struktur model yang lebih kompleks dibandingkan dengan metode tradisional, yang dilatih pada sejumlah besar data untuk mendapatkan kinerja yang relatif lebih baik [14]. Secara umum, kinerja dapat ditingkatkan dengan meningkatkan kedalaman dan lebarnya [14]. Namun, ini akan menyebabkan peningkatan parameter, yang dapat menyebabkan peningkatan perhitungan dan *overfitting* [14].

Gagasan utama *Inception* adalah meningkatkan akurasi dan mengurangi kompleksitas komputasi. Ciri utama arsitektur ini adalah peningkatan pemanfaatan sumber daya komputasi di dalam jaringan [15]. Hal ini dicapai dengan desain yang dibuat dengan hati-hati yang memungkinkan untuk meningkatkan kedalaman dan lebar jaringan sambil menjaga jumlah komputasi [15]. Untuk mengoptimalkan kualitas, arsitektur ini melakukan pemrosesan multiskala dalam sebuah blok yang disebut blok *Inception*. Oleh karena itu, blok *Inception* dapat menangkap lebih banyak informasi tanpa meningkatkan kompleksitas jaringan. Pemrosesan multiskala pada blok *Inception* dapat dilihat pada **gambar 2.16**.



Gambar 2.16 Blok *Inception*

2.6 *Tensorflow*

Tensorflow adalah pustaka perangkat lunak sumber terbuka yang dirilis pada 2015 oleh Google untuk memudahkan pengembang

merancang, membuat, dan melatih model *deep learning* [8]. Meskipun *Tensorflow* hanya satu dari beberapa opsi yang tersedia untuk pengembang, *Tensorflow* memiliki fitur yang lengkap dan mudah untuk digunakan [8].

Tensorflow adalah pustaka Python yang memungkinkan pengguna untuk merepresentasikan perhitungan sebagai grafik aliran data. *Node* dalam grafik ini merepresentasikan operasi matematika, sedangkan *edge* mewakili data yang dikomunikasikan dari satu *node* ke *node* lainnya. Data dalam *Tensorflow* direpresentasikan sebagai tensor, yang merupakan array multidimensi (mewakili vektor dengan tensor 1D, matriks dengan tensor 2D, dll.) [8]. *Tensorflow* juga menyertakan Tensorboard, alat untuk visualisasi data. Adapun logo dari *tensorflow* dapat dilihat pada **gambar 2.17**.

Gambar 2.17 Logo *Tensorflow* **[8]**

2.6.1 *Tensorflow Object Detection API*

Tensorflow menciptakan model pembelajaran mesin yang akurat yang mampu menentukan lokasi objek dan mengidentifikasi banyak objek dalam satu gambar tetap, yang menjadi tantangan utama dalam *computer vision*. *Tensorflow Object Detection API* adalah kerangka kerja sumber terbuka yang dibangun di atas *Tensorflow* yang membuatnya mudah untuk membangun, melatih, dan menggunakan model deteksi objek [8].

2.7 *OpenCV*

OpenCV (Open Source Computer Vision Library) adalah pustaka perangkat lunak sumber terbuka untuk aplikasi *computer vision* dan pembelajaran mesin. *OpenCV* tersedia dalam berbagai macam bahasa pemrograman termasuk python dan C++. Ada banyak fungsi pengolahan citra digital yang tersedia pada pustaka ini yang dapat sangat bermanfaat

[16]. Pustaka ini memiliki lebih dari 2500 algoritma yang sudah dioptimalkan, yang mencakup *computer vision* dan pembelajaran mesin yang mutakhir [16]. Algoritma ini dapat digunakan untuk mendeteksi dan mengenali wajah, mengidentifikasi objek, mengklasifikasikan tindakan manusia dalam video, melacak pergerakan kamera, melacak objek bergerak, mengekstraksi model objek 3D, menyatukan citra bersama untuk menghasilkan citra resolusi tinggi, menemukan citra serupa dari basis data, menghapus mata merah dari citra yang diambil menggunakan flash, mengikuti gerakan mata, mengenali pemandangan, dll [16].

2.8 Google Colaboratory

Google Colaboratory atau *Google Colab* adalah layanan *cloud* gratis dan sekarang juga sudah mendukung GPU gratis. *Google Colab* menggunakan *Jupyter Notebook* yang tidak lagi memerlukan pengaturan dan sepenuhnya berjalan di cloud. *Google Colab* dapat digunakan menulis dan mengeksekusi kode, menyimpan dan membagikan analisis, dan mengakses sumber daya komputasi yang kuat, semuanya gratis dari browser [17].

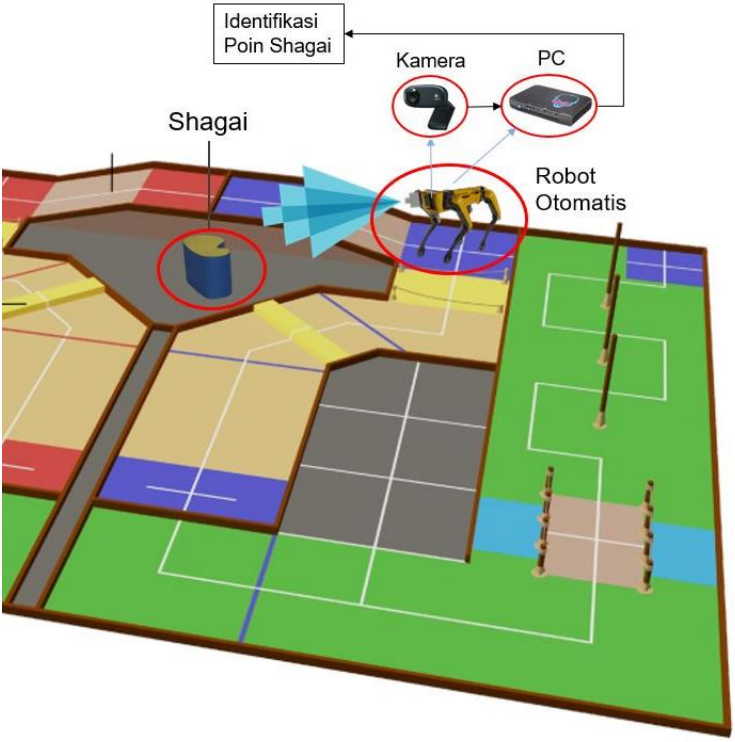
2.9 Kamera Logitech C270

Kamera Logitech C270 merupakan sebuah kamera USB. Kamera ini mengambil video untuk masukan pada komputer. Untuk pengambilan video, kamera ini memiliki tiga resolusi yaitu 320x240, 640x480, dan 800x600 [17]. Kamera ini memiliki kecepatan maksimal 30 fps (*frame per second*) [18]. Tampilan luar dari kamera ini dapat dilihat pada **gambar 2.18**.

Gambar 2.18 Kamera Logitech C270 **[18]**

BAB III
PERANCANGAN SISTEM

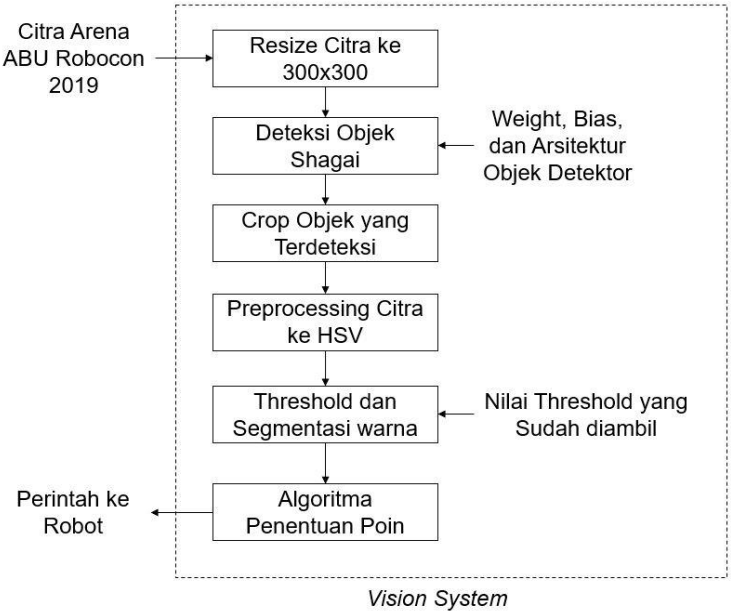
Bab ini membahas tentang perancangan *vision system* untuk identifikasi *Shagai* pada ABU Robocon 2019 secara keseluruhan. Skema keseluruhan sistem ditunjukkan pada **gambar 3.1**. Sistem ini dirancang bertujuan untuk bisa mengidentifikasi poin dari Shagai pada pertandingan ABU Robocon 2019. Sistem terdiri dari kamera dan komputer, yang terletak pada robot otomatis. Kamera digunakan untuk mengambil citra *Shagai* saat robot otomatis berada di area *Mountain Urtuu* dan komputer digunakan untuk mengolah data tersebut sehingga bisa mengidentifikasi poin yang sudah didapatkan. Setelah poin yang didapatkan lebih dari sama dengan 50, sistem mengirim perintah ke robot otomatis untuk berjalan hingga *finish*.



Gambar 3.1 Skema Keseluruhan Sistem

3.1 Diagram Blok Sistem

Diagram blok sistem ditunjukkan pada **gambar 3.2**.



Gambar 3.2 Diagram Blok Sistem

Pada tugas akhir ini *input* didapat dari kamera dengan resolusi yaitu 640x480 piksel. Setelah itu citra diolah pada komputer mengunakan pustaka OpenCV. Proses pengolahan citra ini nantinya bertujuan untuk mengidentifikasi poin dari *Shagai*. Setelah didapatkan total poinnya, sistem akan mengirim perintah ke robot berupa data serial dengan nilai 0 (untuk tetap berhenti) jika poin kurang dari 50 dan 1 (untuk jalan) jika lebih dari sama dengan 50.

3.2 Perancangan Mekanik

Perancangan mekanik merupakan langkah paling awal dalam keseluruhan perancangan sistem pada tugas akhir ini. Perancangan mekanik dilakukan agar sistem yang akan dibuat bisa sesuai dengan kondisi nyata saat berada pada pertandingan. Tinggi kamera tidak boleh melebihi satu meter, karena merupakan tinggi robot maksimal.

Pada tugas akhir ini peneliti memilih tinggi kamera sekitar 80 cm dari tanah. Sedangkan untuk sudutnya dipilih 20 derajat. Pemilihan

didasarkan pada hasil citra yang didapat pada tinggi dan sudut tersebut sanggup mencakup objek *Shagai* pada jarak dekat dan jauh dalam lingkup tempat mendarat *Shagai*.

3.3 Pelatihan Objek Detektor *Shagai*

Pelatihan objek detektor *Shagai* dilakukan untuk mendapatkan suatu model yang bisa mendeteksi objek *Shagai*. Pada tugas akhir ini pelatihan dilakukan dengan cara *Fine Tuning* atau *Transfer Learning*, jadi pelatihan tidak benar-benar dimulai dari awal, melainkan mengambil model yang sudah ada atau yang biasa disebut *pre-trained model*. Untuk menyesuaikan dengan kelas dan *dataset* yang telah peneliti buat, maka dilakukanlah *Transfer Learning* dengan mengganti layer paling terakhir sesuai dengan jumlah kelas yang akan dilatihkan. Pada tugas akhir ini, objek detektor yang digunakan adalah SSD dan arsitektur modelnya menggunakan *Inception*. Arsitektur SSD dan *Inception* telah dijelaskan pada Bab II.

3.3.1 Pemilihan *Pre-trained Model*

Selain menyediakan program untuk pelatihan objek detektor, pengembang *Tensorflow Object Detection API* juga telah menyediakan pre-trained model yang bisa diunduh secara gratis. Beberapa pre-trained model dapat dilihat pada **gambar 3.3**. Pada tugas akhir ini, peneliti memilih menggunakan ssd_inception_v2_coco karena memiliki akurasi deteksi yang cukup tinggi dan kecepatan eksekusi yang lumayan cepat.

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v1_0.75_depth_coco ☆	26	18	Boxes
ssd_mobilenet_v1_quantized_coco ☆	29	18	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco ☆	29	16	Boxes
ssd_mobilenet_v1_ppn_coco ☆	26	20	Boxes
ssd_mobilenet_v1_fpn_coco ☆	56	32	Boxes
ssd_resnet_50_fpn_coco ☆	76	35	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssd_mobilenet_v2_quantized_coco	29	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes

Gambar 3.3 *Pre-trained Model*

3.3.2 Konfigurasi *Training Pipeline*

Konfigurasi *training pipeline* bertujuan untuk mengatur kebutuhan untuk pelatihan, seperti mengatur model mana yang akan dipakai, berapa kelas yang akan dilatihkan, berkas mana yang digunakan untuk pelatihan, berkas mana yang digunakan untuk validasi, banyaknya *epoch*, dan lain-lain. Konfigurasi training pipeline dapat dilihat pada **gambar 3.4**.

pipeline.config

```
model {
  ssd {
    num_classes: 1
    image_resizer {
      fixed_shape_resizer {
        height: 300
        width: 300
      }
    }
    feature_extractor {
      type: "ssd_inception_v2"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 3.9999998989515007e-05
          }
        }
        initializer {
          truncated_normal_initializer {
            mean: 0.0
            stddev: 0.029999999329447746
          }
        }
```

Gambar 3.4 Konfigurasi *Training Pipeline*

3.3.3 Tahap Pelatihan

Ketika training pipeline telah selesai dikonfigurasi, tahap selanjutnya adalah tahap pelatihan. Pada tahap pelatihan ini, untuk mempercepat proses pelatihan, peneliti menggunakan *Google*

Colaboratory atau biasa disebut *Google Colab*. *Google Colab* adalah layanan *cloud* gratis dan sekarang juga sudah mendukung GPU gratis. Peneliti lebih memilih menggunakan *Google Colab* untuk eksekusi program pelatihan daripada menggunakan perangkat laptop milik sendiri, hal ini dilakukan karena *Google Colab* memiliki spesifikasi perangkat keras yang jauh lebih tinggi sehingga program pelatihan dapat berjalan dengan jauh lebih cepat.

Program pelatihan telah disediakan oleh pengembang *Tensorflow Object Detection API*. Hal yang perlu dilakukan adalah menentukan *path* berkas *training pipeline* saat mengeksekusi program tersebut pada *command line*. Hasil eksekusi program pelatihan pada *Google Colab* dapat dilihat pada **gambar 3.5**.

Gambar 3.5 Hasil Eksekusi Program Pelatihan pada *Google Colab*

3.3.4 Export Graph Model

Selama proses pelatihan, *Tensorflow* secara otomatis akan membuat berkas *checkpoint*. Berkas ini berjumlah tiga seperti pada **gambar 3.6**. Berkas dengan format ".data" menyimpan variabel *weight* dan *bias*, berkas dengan format ".meta" menyimpan desain jaringan, sedangkan berkas dengan format ".index" menyimpan daftar nama dan bentuk dari variable-variabel pada jaringan. Agar model bisa digunakan untuk *inference*, maka dilakukan *export graph model* untuk menyatukan ketiga berkas tadi dalam satu berkas. Berkas

tersebut akan disimpan dalam format ".pb" atau disebut *protocol buffer* (protobuf).

Gambar 3.6 Berkas *Checkpoint*

Export graph model dilakukan dengan cara mengeksekusi program export_inference_graph.py, program tersebut telah tersedia pada *Tensorflow Object Detection API*.

3.3.5 Membuat *Text Graph* untuk *OpenCV*

Hasil berkas dari *export graph model* sebenarnya sudah bisa peneliti pakai untuk menjalankan deteksi objek, namun deteksi hanya bisa dilakukan pada program python dengan mengimpor pustaka *Tensorflow*. Padahal disini peneliti butuh agar program berjalan pada Bahasa C++ menggunakan *OpenCV*, oleh karena itu model yang telah dibuat sebelumnya dikonfigurasi lagi agar bisa berjalan pada *OpenCV* C++. *OpenCV* memerlukan berkas konfigurasi tambahan untuk mengimpor model deteksi objek dari *Tensorflow*. Berkas ini didasarkan pada versi teks dari hasil *export graph model* yang tersedia dalam format ".pb". Berkas *graph* versi teks ini memiliki format ".pbtxt" atau disebut *protocol buffer text*. Untuk membuat berkas ini, peneliti menggunakan program dari *OpenCV* 4.0.0 yaitu tf_text_graph_ssd.py.

3.4 *Vision System*

Pada bagian vision system ini akan dibuat perancangan algoritma pemrograman. Di dalamnya terdapat beberapa proses yaitu *resize* citra, deteksi objek *Shagai*, crop objek yang terdeteksi, *preprocessing* citra ke HSV, *threshold* dan segmentasi warna, lalu yang terakhir ada algoritma penentuan poin *Shagai*. Proses tersebut bertujuan untuk mengidentifikasi poin dari *Shagai*. Setelah didapatkan total poinnya, sistem akan mengirim perintah ke robot berupa data serial dengan nilai 0 (untuk tetap berhenti) jika poin kurang dari 50 dan 1 (untuk jalan) jika lebih dari sama dengan 50.

3.4.1 *Resize* Citra

Resize citra dilakukan menggunakan fungsi pada pustaka *OpenCV*. Proses ini bertujuan untuk merubah citra dari ukuran

640x480 ke 300x300. Hal ini dilakukan karena nantinya model objek detektor *Shagai* menerima *input* citra dengan ukuran 300x300. Hasil *resize* citra ditunjukkan pada **gambar 3.7**.

Gambar 3.7 *Resize* Citra dari Ukuran 640x480 ke 300x300

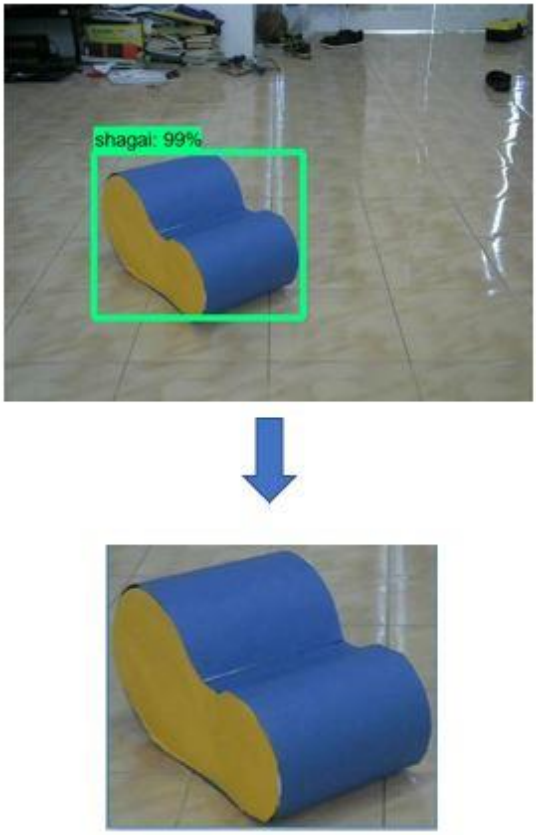
3.4.2 Deteksi Objek *Shagai*

Proses deteksi objek *Shagai* dilakukan bertujuan untuk mendeteksi semua *Shagai* yang ada pada citra. Setelah citra diubah ukurannya ke 300x300 piksel, kemudian citra tersebut dijadikan *input* untuk menjalankan deteksi *Shagai* menggunakan model yang telah berhasil dilatih sebelumnya. Model ini berisi *weight*, *bias*, dan arsitektur jaringan dari hasil pelatihan menggunakan *Tensorflow Object Detection API*. Hasil deteksi Shagai ditunjukkan pada **gambar 3.8**. *Output* dari deteksi *Shagai* yaitu berupa label objek dan juga *bounding box*. *Bounding box* memiliki nilai koordinat x, y serta panjang dan lebarnya. Nilai inilah nantinya yang akan digunakan pada proses *crop*.

Gambar 3.8 Hasil Deteksi *Shagai*

3.4.3 Crop Objek yang Terdeteksi

Proses *crop* objek ini dilakukan bertujuan untuk memudahkan proses pengolahan citra. Proses ini dilakukan menggunakan fungsi yang sudah disediakan pada pustaka *OpenCV*. Proses *crop* dilakukan sesuai dengan nilai koordinat *bounding box* yang telah didapatkan pada proses sebelumnya. Sehingga nantinya akan didapatkan citra persegi panjang yang baru sesuai dengan ukuran *bounding box* yang terdeteksi. Nantinya objek yang sudah melalui proses *crop* ini akan diolah lagi untuk diambil informasi warnanya. Ilustrasi proses *crop* objek yang terdeteksi ini ditunjukkan pada **gambar 3.9**.



Gambar 3.9 Ilustrasi *Crop* Objek

3.4.4 *Preprocessing* Citra ke HSV

Preprocessing citra ke ruang warna HSV dilakukan untuk mendapatkan segmentasi warna yang lebih baik daripada menggunakan ruang warna RGB. Deskripsi warna dalam ruang HSV lebih relevan, karena ruang warna HSV memisahkan komponen warna dan cahaya, sehingga lebih cocok digunakan untuk penerapan segmentasi warna nantinya. Proses ini dilakukan pada citra objek yang sebelumnya telah melalui proses *crop*.

Pustaka *OpenCV* telah menyediakan fungsi untuk mengubah citra dari ruang warna RGB ke HSV. Program untuk mengubah warna

citra dari ruang warna RGB ke HSV ditunjukkan pada **gambar 3.10**. Objek frame digunakan sebagai wadah dari citra yang kita muat. Lalu objek hsv digunakan untuk wadah dari citra dari objek frame yang diubah ke ruang warna HSV.

```
Mat frame;
Mat hsv;

frame = imread("gambar.jpg", IMREAD_COLOR);
cvtColor(frame, hsv, COLOR_BGR2HSV);
```

Gambar 3.10 Mengubah Ruang Warna RGB ke HSV

3.4.5 Pengambilan *Threshold* HSV

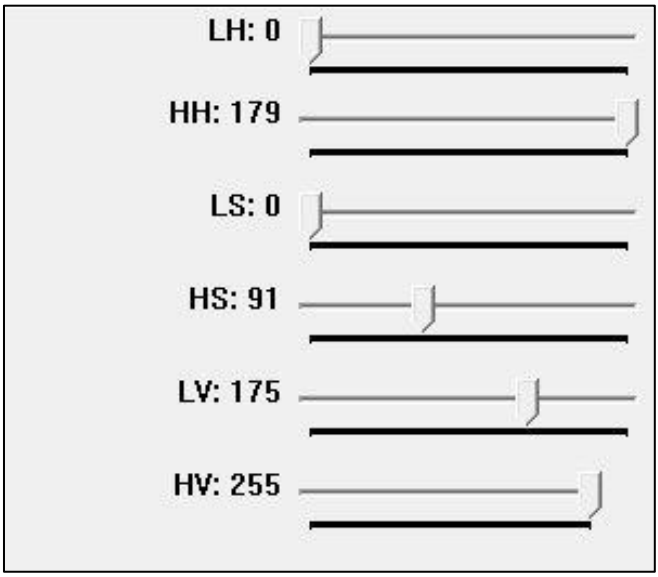
Pengambilan *threshold* HSV bertujuan untuk mendapatkan nilai *threshold* warna HSV dan nantinya bisa membedakan warna dari masing-masing sisi *Shagai*, yaitu warna merah/biru, warna emas, dan warna perak. Pengambilan *threshold* dilakukan dengan mengambil dua nilai, yaitu nilai *threshold* atas dan nilai *threshold* bawah. Nantinya, warna yang terdeteksi adalah warna yang nilai masing-masing H, S, dan V berada di antara kedua nilai *threshold* tersebut. Pengambilan *threshold* ini dilakukan dengan cara manual.

Program menggunakan pustaka *OpenCV* digunakan peneliti untuk mempermudah mengambil nilai *threshold*. Mula-mula dibuat sebuah *trackbar* untuk masing-masing nilai *threshold* atas dan bawah komponen H, S, dan V. Program untuk membuat *trackbar* dapat dilihat pada **gambar 3.11** dan hasil pembuatan jendela *trackbar* dapat dilihat pada **gambar 3.12**. LowH merupakan *threshold* bawah komponen H, sedangkan highH merupakan *threshold* atas komponen H. Begitu pula untuk komponen S dan komponen V.

```
int hmin = 0, hmax = 179;
int smin = 0, smax = 91;
int vmin = 175, vmax  = 255;

namedWindow("trackbar", WINDOW_AUTOSIZE);
createTrackbar("LH", "trackbar", &hmin, 179);
createTrackbar("HH", "trackbar", &hmax, 179);
createTrackbar("LS", "trackbar", &smin, 255);
createTrackbar("HS", "trackbar", &smax, 255);
createTrackbar("LV", "trackbar", &vmin, 255);
createTrackbar("HV", "trackbar", &vmax, 255);
```

Gambar 3.11 Program untuk Membuat *Trackbar*



Gambar 3.12 Jendela *Trackbar*

Setelah *trackbar* dibuat, citra diambil dari objek *Shagai*, lalu diubah ke ruang warna HSV. Program untuk penerapan *threshold* pada citra HSV ditunjukkan pada **gambar 3.13**.

```
Mat frame;
Mat hsv;
Mat thr;

frame = imread("gambar.jpg", IMREAD_COLOR);
cvtColor(frame, hsv, COLOR_BGR2HSV);
inRange(hsv, Scalar(0, 0, 0), Scalar(179, 255,255), thr);
```

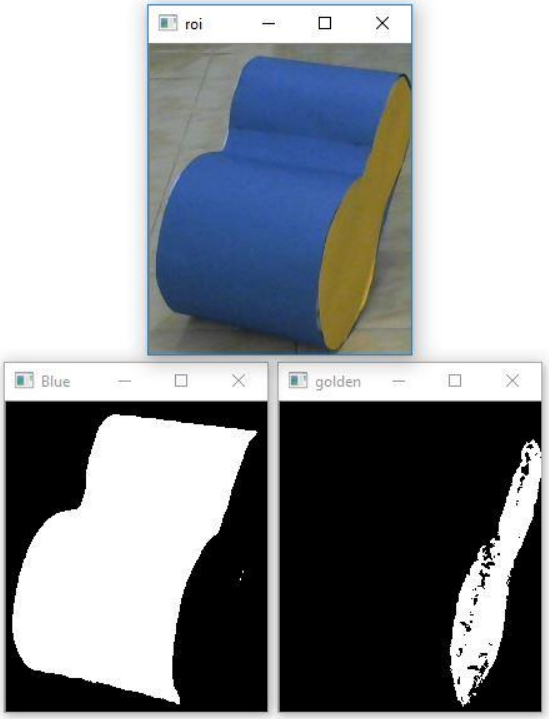
Gambar 3.13 Program untuk Menerapkan *Threshold*

3.4.6 *Threshold* dan Segmentasi Warna *Shagai*

Segmentasi warna *Shagai* bertujuan untuk memisahkan warna merah/biru, warna emas, dan warna perak pada objek *Shagai*. Masing-masing objek *Shagai* yang telah melalui proses *crop* diubah ruang warnanya menjadi HSV. Segmentasi dilakukan dengan menerapkan *threshold* pada citra objek dengan ruang warna HSV menggunakan nilai yang telah didapat pada proses sebelumnya, lalu fungsi *contour* pada *OpenCV* digunakan untuk mencari semua kontur yang

memungkinkan dari citra biner hasil segmentasi warna. Dari semua kontur yang terdeteksi, dipilih kontur yang memiliki luas paling besar.

Untuk visualisasi hasil *threshold* dan segmentasi warna, maka citra hasil proses tersebut ditampilkan dalam sebuah jendela yang ditunjukkan pada **gambar 3.14**. Terlihat bahwa sisi *Shagai* dengan warna biru tersegmentasi pada jendela "Blue" dan sisi *Shagai* dengan warna emas tersegmentasi pada jendela "Golden".



Gambar 3.14 Citra Hasil *Threshold* dan Segmentasi

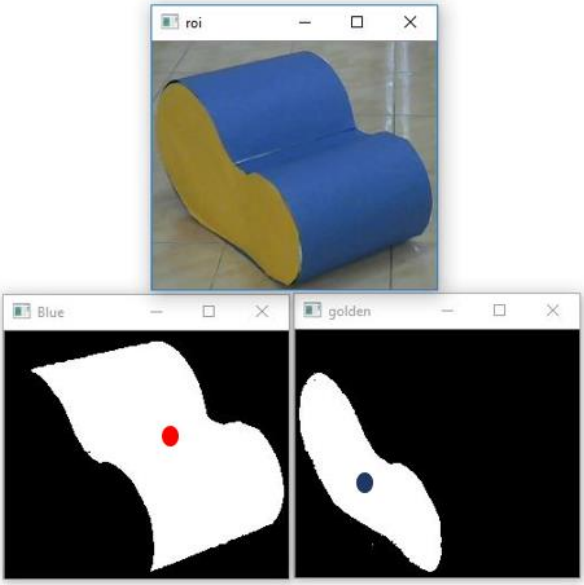
3.4.7 Algoritma Penentuan Poin *Shagai*

Inti dari keseluruhan sistem pada tugas akhir ini adalah mengidentifikasi warna sisi atas pada objek *Shagai* untuk bisa mengetahui poinnya. Untuk melakukan hal tersebut, peneliti menggunakan metode membandingkan koordinat y dari warna selimut *Shagai* (warna merah/biru) dengan koordinat y dari salah satu warna lain yang terdeteksi yaitu perak atau emas. Pada pustaka *OpenCV*,

koordinat x,y (0,0) dimulai dari pojok kiri atas, artinya nilai koordinat x semakin besar apabila posisinya semakin ke kanan dan nilai koordinat y semakin besar apabila posisinya semakin ke bawah.

Karena objek berbentuk tiga dimensi dan berbentuk unik, jika direpresentasikan dalam ruang dua dimensi maka hanya akan ada dua warna yang terdeteksi. Dari kedua warna yang terdeteksi tersebut, bisa dilakukan identifikasi untuk mengetahui warna sisi atasnya. Ambil contoh jika selimut *Shagai* berwarna biru. Jika warna biru dan warna emas terdeteksi, dan titik tengah koordinat y dari kontur warna emas lebih kecil daripada warna biru, maka sisi atas *Shagai* adalah warna emas. Jika titik tengah koordinat y kontur warna biru lebih kecil daripada warna emas, maka sisi atas berwarna biru. Begitu juga hal tersebut berlaku jika yang terdeteksi adalah warna biru dan perak.

Ilustrasi penentuan poin ditunjukkan pada **gambar 3.15**. Terlihat bahwa koordinat y dari titik tengah sisi warna biru (titik berwarna merah) lebih di atas daripada koordinat y dari titik tengah sisi warna emas (titik berwarna biru). Maka dapat disimpulkan bahwa warna bagian atasnya adalah biru.



Gambar 3.15 Ilustrasi Penentuan Poin *Shagai*

Setelah warna sisi atas *Shagai* berhasil diketahui, maka dengan sedikit algortima kondisi, poin yang didapat juga bisa diketahui. Poin bernilai 50 untuk *Shagai* dengan sisi atas berwarna emas, 40 untuk warna perak, dan 20 untuk warna selimut (merah/biru). Hasil ini nantinya akan digunakan untuk menentukan langkah dari robot otomatis. Jika poin lebih dari sama dengan 50, maka sistem akan mengirim perintah agar robot berjalan hingga *finish*, dan jika poin kurang dari 50 maka robot akan tetap menunggu untuk lemparan selanjutnya.

BAB IV
PENGUJIAN DAN ANALISIS

Bab ini membahas tentang pengujian dan analisis dari sistem yang telah dirancang dan dibuat pada tugas akhir ini, yaitu *vision system* untuk identifikasi *Shagai* pada ABU Robocon 2019. Tujuan dari bab ini adalah untuk mengetahui apakah sistem dapat menyelesaikan permasalahan yang telah dirumuskan atau tidak. Pengujian dan analisis yang dilakukan meliputi pengujian detektor objek *Shagai*, pengujian segmentasi warna *Shagai*, dan pengujian deteksi pada beberapa *Shagai* dalam satu gambar.

4.1 Pengujian Detektor Objek *Shagai*

Pengujian detektor objek *Shagai* dilakukan untuk menentukan seberapa akurat sistem dapat mendeteksi objek *Shagai* beserta lokasinya pada citra. Pengujian dilakukan pada dua macam kondisi untuk tiap *Shagai* merah dan biru, yaitu berdasarkan jarak yang berbeda-beda dan berdasarkan hadap sisi yang berbeda-beda. Hasil pengujian akan diukur seberapa akurat menggunakan *metric f-measure*.

$$Precision = \frac{\text{TP}}{\text{TP+FP}} = \frac{\text{TP}}{\text{all } detections} \tag{3.1}$$

$$Recall = \frac{\text{TP}}{\text{TP+FN}} = \frac{\text{TP}}{\text{all ground truths}} \tag{3.2}$$

$$F - Score = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.3}$$

Dimana:

TP : deteksi yang benar, yaitu deteksi dengan IOU ≥ 0.5.

FP : deteksi yang salah yaitu deteksi dengan IOU < 0.5 atau satu objek terdeteksi dua kali.

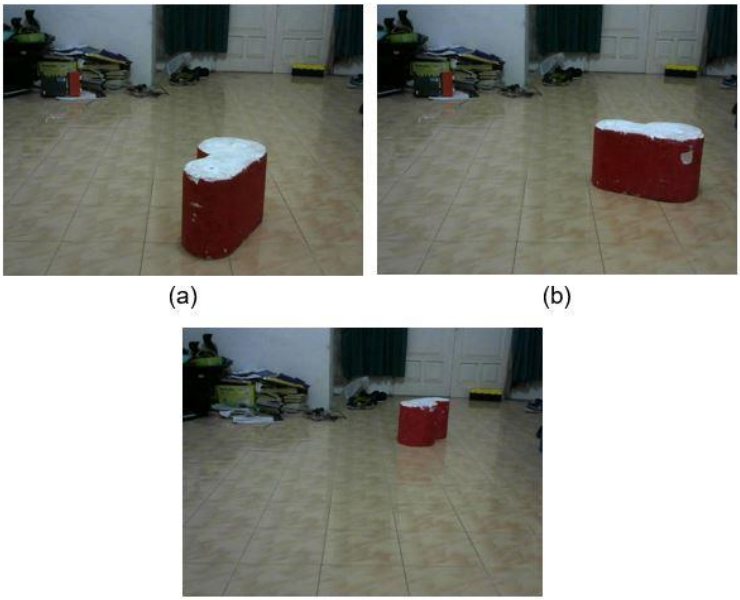
FN : Ada objek tapi tidak terdeteksi.

4.1.1 Pengujian Berdasarkan Jarak

Pengujian berdasarkan jarak dilakukan pada tiga variasi jarak dari posisi kamera, yaitu jarak 90 cm – 180 cm (jarak dekat), jarak 181 cm – 270 cm (jarak menengah), dan jarak 271 cm – 360 cm (jarak jauh). Pada pengujian ini, tiap variasi jarak diuji menggunakan 15 data. Data tersebut diambil terlebih dahulu menggunakan kamera, data ini dipisahkan dan tidak boleh data yang sama yang dipakai untuk pelatihan. Contoh data yang digunakan untuk pengujian dapat dilihat

pada **gambar 4.1** untuk *Shagai* merah dan **gambar 4.2** untuk *Shagai* biru.

Pengujian dilakukan dengan cara menjalankan program deteksi menggunakan model yang telah didapatkan dari hasil pelatihan. Citra hasil deteksi ditampilkan pada sebuah jendela, lalu hasilnya dibandingkan dengan hasil deteksi yang seharusnya. Hasil pengujian ditunjukkan pada **tabel 4.1**. Contoh hasil deteksi objek *Shagai* yang benar ditunjukkan pada **gambar 4.3**. Sedangkan untuk contoh hasil deteksi objek *Shagai* yang salah ditunjukkan pada **gambar 4.4**. Dari hasil pengujian, didapatkan bahwa detektor objek *Shagai* memiliki akurasi total *F-Score* sebesar 97,8% untuk *Shagai* merah dan 98,9% untuk *Shagai* biru. Dari hasil pengujian pada **tabel 4.1**, menunjukkan bahwa jarak mempengaruhi deteksi. Jarak yang diujikan sudah sesuai dengan kisaran jarak tempat mendarat *Shagai* pada arena pertandingan.



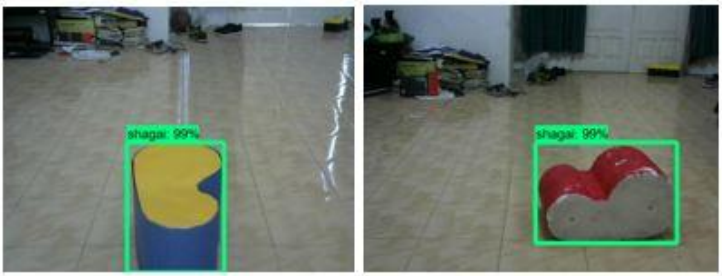
Gambar 4.1 Contoh Data Pengujian *Shagai* Merah (a) Jarak Dekat (b) Jarak Menengah (c) Jarak Jauh

(a)

(b)

(c)

Gambar 4.2 Contoh Data Pengujian *Shagai* Biru (a) Jarak Dekat (b) Jarak Menengah (c) Jarak Jauh



Gambar 4.3 Contoh Hasil Deteksi *Shagai* yang Benar

Gambar 4.4 Contoh Hasil Deteksi *Shagai* yang Salah

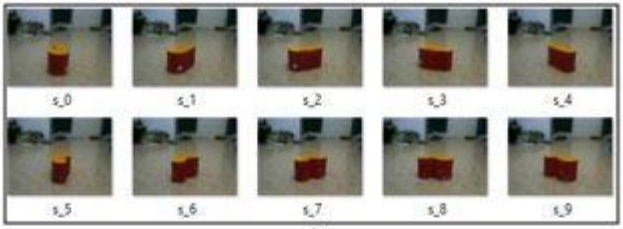
Tabel 4.1 Hasil Pengujian Detektor Objek *Shagai* Berdasarkan Jarak

Warna	Jarak	TP	FP	FN	P	R	F-Score
Merah	Dekat	15	0	0	1.0	1.0	100%
	Menengah	15	0	0	1.0	1.0	100%
	Jauh	14	1	1	0.933	0.933	93,3%
	Total	44	1	1	0.978	0.978	97,8%
Biru	Dekat	15	0	0	1.0	1.0	100%
	Menengah	15	0	0	1.0	1.0	100%
	Jauh	14	0	1	1.0	0.933	96,5%
	Total	44	0	1	1.0	0.978	98,9%

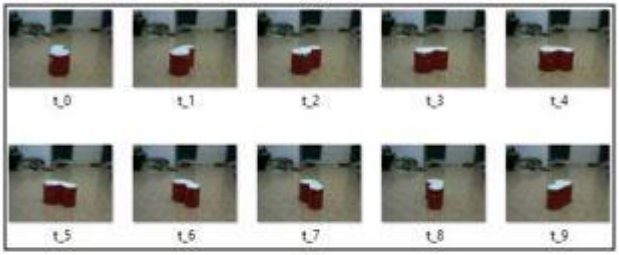
4.1.2 Pengujian Berdasarkan Hadap Sisi

Pengujian berdasarkan hadap sisi dilakukan pada empat variasi sisi *Shagai*, yaitu sisi *Shagai* saat permukaan atasnya berwarna emas (poin 50), saat permukaan atasnya berwarna perak (poin 40), saat permukaan atasnya berwarna biru/merah atau peneliti sebut sisi selimut (poin 20), dan saat sisi *Shagai* miring (poin 20). Pada pengujian ini, tiap variasi hadap sisi diuji menggunakan 15 data pada sudut yang berbeda-beda. Data tersebut diambil terlebih dahulu menggunakan kamera, data ini dipisahkan dan tidak boleh data yang sama yang dipakai untuk pelatihan. Contoh data yang digunakan untuk pengujian dapat dilihat pada **gambar 4.5** untuk *Shagai* merah dan **gambar 4.6** untuk *Shagai* biru.

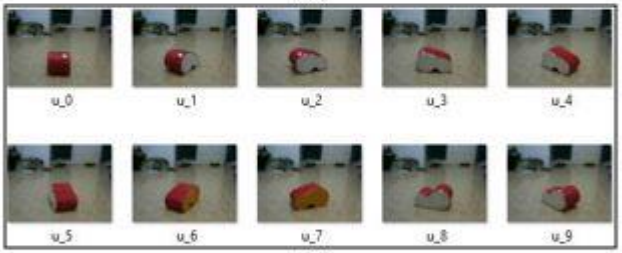
Pengujian dilakukan dengan cara menjalankan program deteksi menggunakan model yang telah didapatkan dari hasil pelatihan. Citra hasil deteksi ditampilkan pada sebuah jendela, lalu hasilnya dibandingkan dengan hasil deteksi yang seharusnya. Hasil pengujian ditunjukkan pada **tabel 4.2**. Contoh hasil deteksi objek *Shagai* yang benar ditunjukkan pada **gambar 4.7**. Sedangkan untuk contoh hasil deteksi objek *Shagai* yang salah ditunjukkan pada **gambar 4.8**. Dari hasil pengujian, didapatkan bahwa detektor objek *Shagai* memiliki akurasi total *F-Score* sebesar 99,2% untuk *Shagai* merah dan 98,4% untuk *Shagai* biru. Hasil ini menunjukkan bahwa hadap sisi *Shagai* tidak terlalu mempengaruhi deteksi.



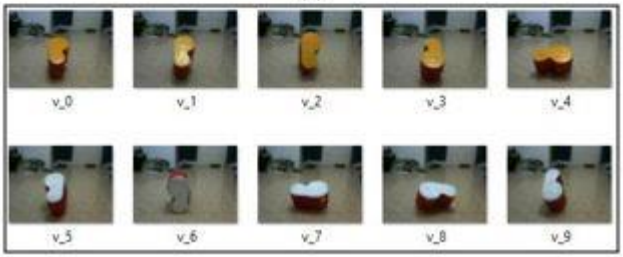
(a)



(b)

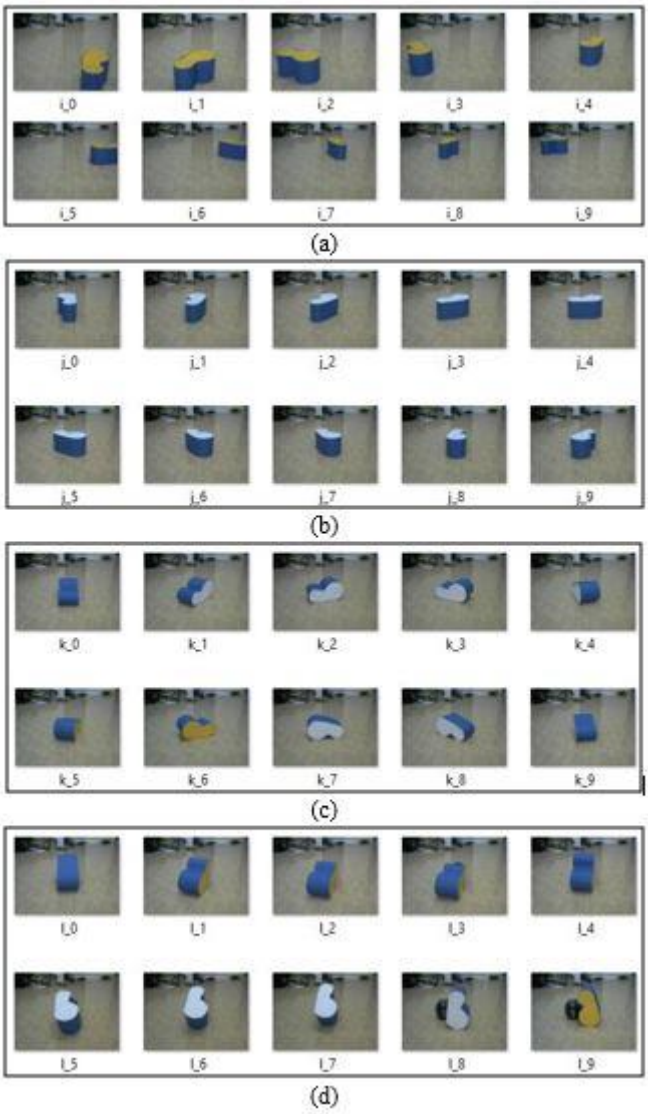


(c)

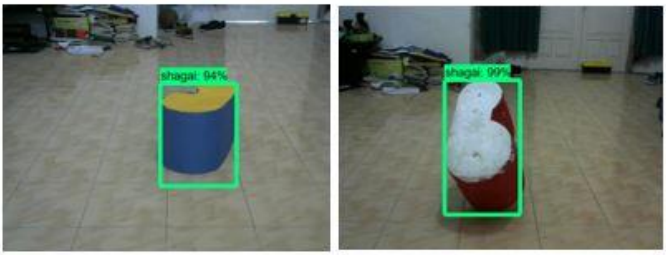


(d)

Gambar 4.5 Contoh Data Pengujian *Shagai* Merah (a) Sisi Emas (b) Sisi Perak (c) Sisi Selimut (d) Sisi Miring



Gambar 4.6 Contoh Data Pengujian *Shagai* Biru (a) Sisi Emas (b) Sisi Perak (c) Sisi Selimut (d) Sisi Miring



Gambar 4.7 Contoh Hasil Deteksi *Shagai* yang Benar pada Pengujian Berdasarkan Hadap Sisi

Gambar 4.8 Contoh Hasil Deteksi *Shagai* yang Salah pada Pengujian Berdasarkan Hadap Sisi

Tabel 4.2 Hasil Pengujian Deteksi *Shagai* Berdasarkan Hadap Sisi

Warna	Sisi	TP	FP	FN	P	R	F-Score
Merah	Emas	15	0	0	1.0	1.0	100%
	Perak	15	0	0	1.0	1.0	100%
	Selimut	15	0	0	1.0	1.0	100%
	Miring	15	1	0	0.937	1.0	96,7%
	Total	60	1	0	0.984	1.0	99,2%
Biru	Emas	15	1	1	0.937	0.937	93,7%
	Perak	15	0	0	1.0	1.0	100%
	Selimut	15	0	0	1.0	1.0	100%
	Miring	15	0	0	1.0	1.0	100%
	Total	60	1	1	0.984	0.984	98,4%

4.2 Pengujian Segmentasi Warna *Shagai*

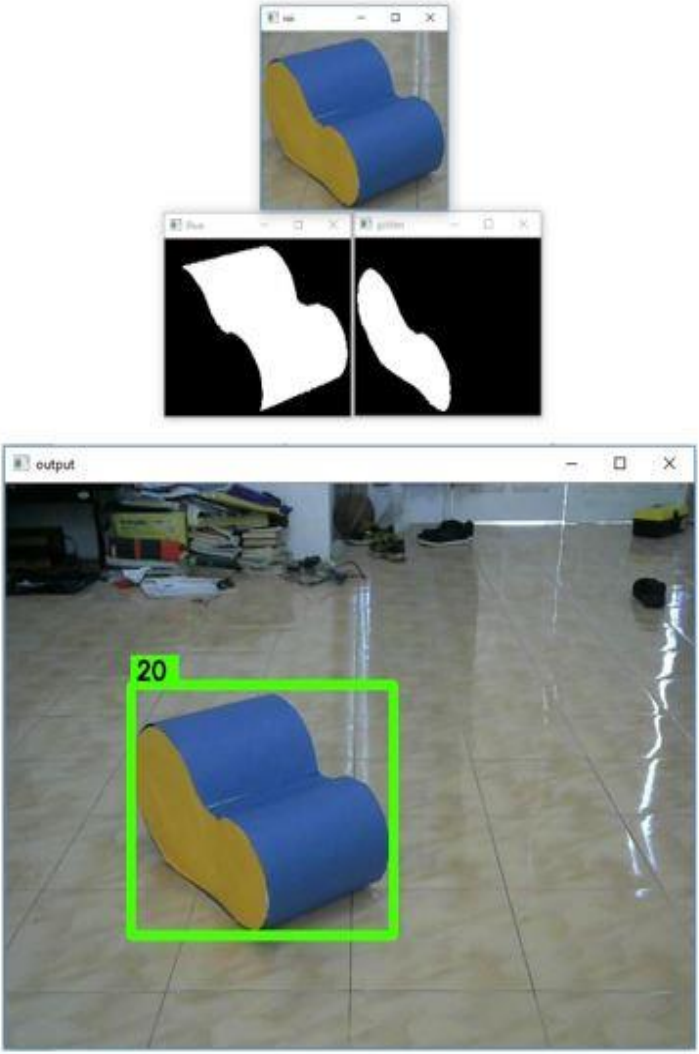
Pengujian segmentasi warna *Shagai* dilakukan untuk mengetahui seberapa akurat algoritma segmentasi warna dalam menentukan jumlah poin tiap *Shagai*. Pengujian dilakukan pada dua macam kondisi untuk tiap *Shagai* merah dan biru, yaitu berdasarkan jarak yang berbeda-beda dan berdasarkan hadap sisi yang berbeda-beda.

4.2.1 Pengujian Berdasarkan Jarak

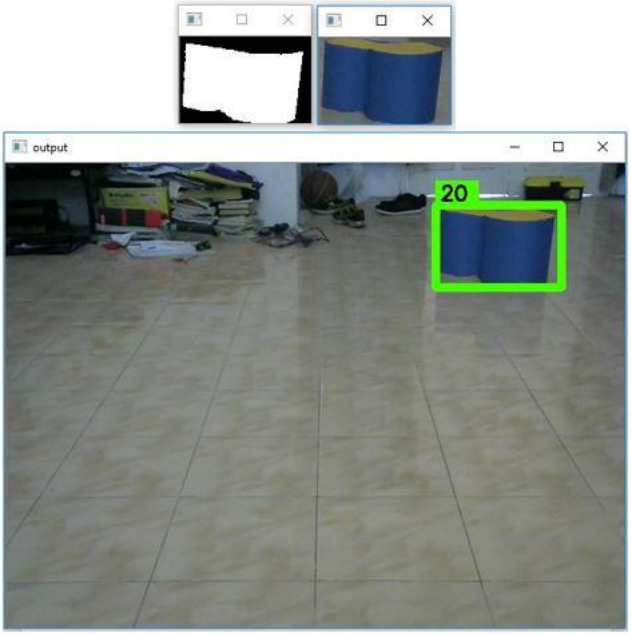
Pengujian berdasarkan jarak dilakukan pada tiga variasi jarak dari posisi kamera, yaitu jarak 90 cm – 180 cm (jarak dekat), jarak 181 cm – 270 cm (jarak menengah), dan jarak 271 cm – 360 cm (jarak jauh). Pada pengujian ini, tiap variasi jarak diuji menggunakan 15 data. Data tersebut diambil terlebih dahulu menggunakan kamera, data ini dipisahkan dan tidak boleh data yang sama yang dipakai untuk pelatihan. Contoh data yang digunakan untuk pengujian dapat dilihat pada **gambar 4.1** untuk *Shagai* merah dan **gambar 4.2** untuk *Shagai* biru.

Setelah citra *Shagai* dari berbagai variasi jarak telah dibuat, pengujian siap dilakukan. Pengujian dilakukan dengan cara menjalankan program deteksi menggunakan model yang telah didapatkan dari hasil pelatihan. Selanjutnya, citra hasil deteksi dipotong berdasarkan lokasi objek *Shagai* yang terdeteksi. Lalu, dilakukan segmentasi warna pada tiap sisi *Shagai* yang terlihat dari sudut pandang kamera. Setelah itu titik tengah dari warna-warna yang terdeteksi dibandingkan untuk mengetahui warna apa yang berada di sisi bagian atas.

Hasil pengujian ditunjukkan pada **tabel 4.3**. Contoh hasil segmentasi warna *Shagai* yang benar ditunjukkan pada **gambar 4.9**. Sedangkan untuk contoh hasil segmentasi warna *Shagai* yang salah ditunjukkan pada **gambar 4.10**. Dari **gambar 4.10** terlihat bahwa kegagalan terjadi karena sistem gagal mengidentifikasi salah satu warna yaitu bagian warna emas. Hal ini terjadi karena bagian warna emas tidak masuk dalam nilai *threshold* yang sudah ditentukan, kemungkinan karena adanya perbedaan intensitas cahaya sehingga nilai threshold yang sudah diambil kurang tepat. Hal ini menyebabkan hanya sisi selimut saja yang terdeteksi dan poin yang dihasilkan 20, padahal seharusnya adalah 50 karena sisi atasnya berwarna emas. Dari hasil pengujian, didapatkan bahwa algoritma segmentasi warna yang digunakan untuk menentukan poin *Shagai* memiliki akurasi total sebesar 95,6% untuk *Shagai* merah dan 93,3% untuk *Shagai* biru.



Gambar 4.9 Contoh Hasil Segmentasi Warna *Shagai* yang Benar



Gambar 4.10 Contoh Hasil Segmentasi Warna *Shagai* yang Salah

Tabel 4.3 Hasil Pengujian Segmentasi Warna *Shagai* Berdasarkan Jarak

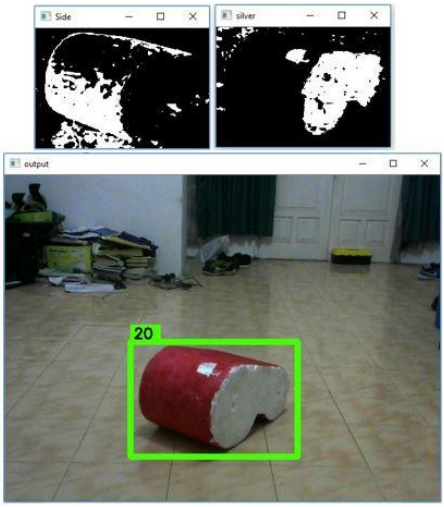
Warna	Jarak	Benar	Salah	Akurasi
Merah	Dekat	15	0	100%
	Menengah	14	1	93,3%
	Jauh	14	1	93,3%
	Total	43	2	95,6%
Biru	Dekat	14	1	93,3%
	Menengah	15	0	100%
	Jauh	13	2	86,7%
	Total	42	3	93,3%

4.2.2 Pengujian Berdasarkan Hadap Sisi

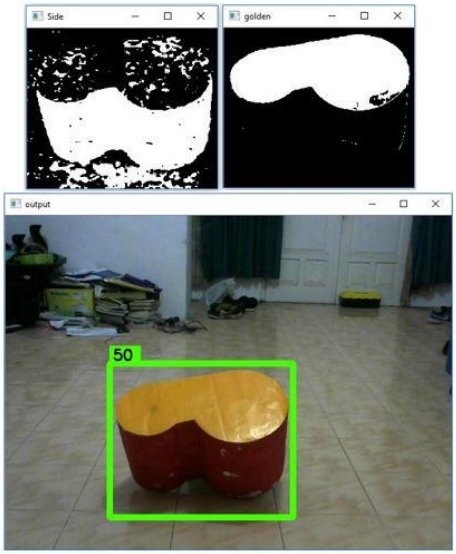
Pengujian berdasarkan hadap sisi dilakukan pada empat variasi sisi *Shagai*, yaitu sisi *Shagai* saat permukaan atasnya berwarna emas (poin 50), saat permukaan atasnya berwarna perak (poin 40), saat permukaan atasnya berwarna biru/merah atau peneliti sebut sisi selimut (poin 20), dan saat sisi *Shagai* miring (poin 20). Pada pengujian ini, tiap variasi hadap sisi diuji menggunakan 15 data pada sudut yang berbeda-beda. Data tersebut diambil terlebih dahulu menggunakan kamera, data ini dipisahkan dan tidak boleh data yang sama yang dipakai untuk pelatihan. Contoh data yang digunakan untuk pengujian dapat dilihat pada **gambar 4.5** untuk *Shagai* merah dan **gambar 4.6** untuk *Shagai* biru.

Setelah citra *Shagai* dari berbagai variasi jarak telah dibuat, pengujian siap dilakukan. Pengujian dilakukan dengan cara menjalankan program deteksi menggunakan model yang telah didapatkan dari hasil pelatihan. Selanjutnya, citra hasil deteksi dipotong berdasarkan lokasi objek *Shagai* yang terdeteksi. Lalu, dilakukan segmentasi warna pada tiap sisi *Shagai* yang terlihat dari sudut pandang kamera. Setelah itu titik tengah dari warna-warna yang terdeteksi dibandingkan untuk mengetahui warna apa yang berada di sisi bagian atas.

Hasil pengujian ditunjukkan pada **tabel 4.4**. Contoh hasil segmentasi warna *Shagai* yang benar ditunjukkan pada **gambar 4.11**. Sedangkan untuk contoh hasil segmentasi warna *Shagai* yang salah ditunjukkan pada **gambar 4.15**. Dari hasil pengujian, didapatkan bahwa segmentasi warna *Shagai* memiliki akurasi total sebesar 86,7% untuk *Shagai* merah dan 88,3% untuk *Shagai* biru. Dari hasil pengujian diketahui bahwa kegagalan terjadi saat sisi *Shagai* miring. Padahal walaupun miring atau tidak, warna sisi bagian atas tetap terlihat sebagai warna sisi tersebut, sedangkan menurut aturan ketika *Shagai* miring / terganjal objek lain maka poin yang didapatkan pasti 20 apapun warna sisi atasnya. Di sini lah sistem tidak dapat menghitung dengan tepat poin yang seharusnya. Hal tersebut yang menjadi kelemahan untuk algoritma penghitungan poin yang peneliti pakai, karena algoritma ini hanya bergantung pada warna sisi atas *Shagai* tanpa bisa mengetahui apakah posisi *Shagai* tersebut miring atau tidak. Namun, kelemahan tersebut bukanlah suatu hal yang beresiko, karena keadaan *Shagai* miring saat mendarat sangat jarang terjadi. Sedangkan untuk kasus selain itu, perhitungan poin berlangsung dengan sangat tepat.



Gambar 4.11 Contoh Hasil Deteksi *Shagai* yang Benar pada Pengujian Berdasarkan Hadap Sisi



Gambar 4.12 Contoh Hasil Deteksi *Shagai* yang Salah pada Pengujian Berdasarkan Hadap Sisi

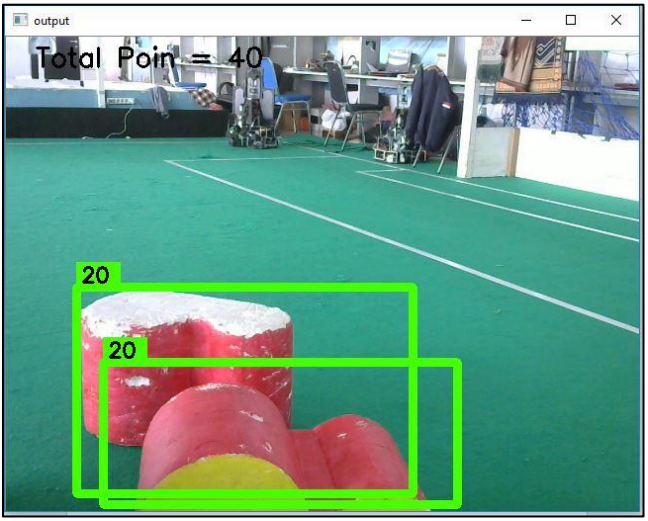
Tabel 4.4 Hasil Pengujian Segmentasi Warna *Shagai* Berdasarkan Hadap Sisi

Warna	Sisi	Benar	Salah	Akurasi
Merah	Emas	15	0	100%
	Perak	15	0	100%
	Selimut	15	0	100%
	Miring	7	8	46,7%
	Total	52	8	86,7%
Biru	Emas	15	0	93,3%
	Perak	15	0	100%
	Selimut	15	0	100%
	Miring	8	7	53,3%
	Total	53	7	88,3%

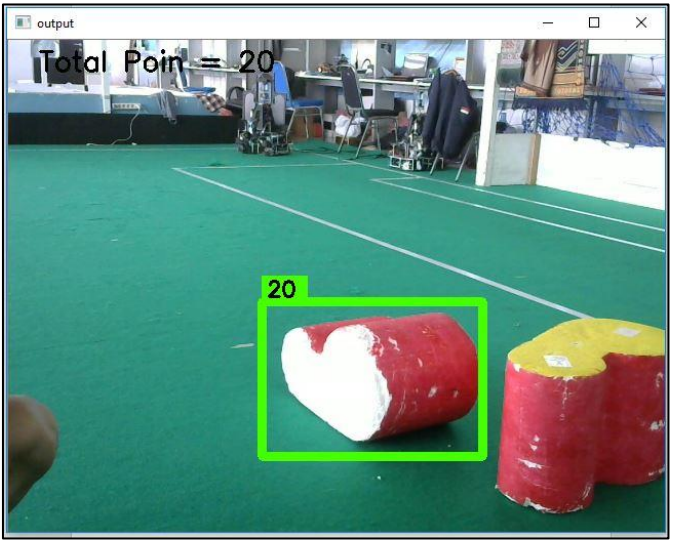
4.3 Pengujian Deteksi Beberapa *Shagai* dalam Satu Gambar

Pengujian ini dilakukan untuk mengetahui seberapa akurat deteksi *Shagai* jika terdapat lebih dari satu *Shagai* di dalamnya. Pengujian dilakukan dengan cara menjalankan program untuk deteksi *Shagai*, lalu menghitung jumlah poin total dari seluruh *Shagai* yang terdeteksi. Pada pengujian ini digunakan 15 data. Pengambilan data dilakukan dengan cara menaruh beberapa Shagai tanpa saling menutupi. Setelah itu diambil gambar mengunakan kamera sebanyak 15 kali dengan beberapa variasi hadap sisi pada masing-masing *Shagai*.

Hasil pengujian dapat dilihat pada **tabel 4.5**. Contoh hasil deteksi yang benar ditunjukkan pada **gambar 4.13**. Sedangkan contoh hasil deteksi yang salah ditunjukkan pada **gambar 4.14**. Dari hasil pengujian didapatkan akurasi sebesar 80%. Dari analisis yang dilakukan, kegagalan terjadi karena kurangnya dataset untuk objek *Shagai* dalam jumlah banyak dalam satu gambar, sehingga menyebabkan model yang didapat dari pelatihan mengalami beberapa kesalahan ketika dihadapkan pada kasus ini.



Gambar 4.13 Contoh Hasil Deteksi Beberapa *Shagai* dalam Satu Gambar yang Benar pada Pengujian



Gambar 4.14 Contoh Hasil Deteksi Beberapa *Shagai* dalam Satu Gambar yang Salah pada Pengujian

Tabel 4.5 Hasil Pengujian Deteksi pada Beberapa Shagai dalam Satu Gambar

Benar	Salah	Akurasi
12	3	80%

4.4 Pengujian Deteksi Berdasarkan Tingkat Pencahayaan

Pengujian ini dilakukan untuk mengetahui seberapa akurat deteksi *Shagai* jika dilakukan pada tingkat pencahayaan yan berbeda-beda. Pada pengujian ini juga dihitung rata-rata *frame per second* (FPS) yang didapat pada pencahayaan tertentu. Pengujian dilakukan pada tiga kondisi yaitu 1 lux (gelap), 18 lux (normal), dan 50 lux (terang). Data pengujian berjumlah 15 tiap variasi kondisinya, dengan kombinasi jarak dan hadap sisi yang berbeda-beda. Hasil pengujian akan diukur seberapa akurat menggunakan *metric f-measure* seperti pada pengujian pertama. Contoh data pengujian ditunjukkan pada **gambar 4.15**.

Hasil yang didapatkan menunjukkan bahwa deteksi terbaik terjadi saat pencahayaan normal (18 lux) dengan nilai F-Score 96,5%, sedangkan saat gelap dan terang banyak objek yang gagal terdeteksi. Hal ini kemungkinan terjadi karena dataset yang digunakan untuk pelatihan model objek detektor *Shagai* diambil hanya saat kondisi pencahayaan normal, sehingga yang dikenali oleh model objek detektor lebih cenderung ke pencahayaan normal. Untuk kecepatan pemrosesan citra didapatkan rata-rata sebesar 2 FPS. Hasil ini ditunjukkan pada **tabel 4.6**.

Gambar 4.15 Contoh Data Pengujian Deteksi dengan Tingkat Pencahayaan (a) Gelap (b) Normal (c) Terang

Tabel 4.6 Hasil Pengujian Deteksi Berdasarkan Tingkat Pencahayaan

Pencahayaan	TP	FP	FN	P	R	F-Score	*Speed*
1 Lux (Gelap)	14	0	1	1.0	0.933	95,6%	1 FPS
18 Lux (Normal)	10	0	5	1.0	0.667	80%	2 FPS
50 Lux (Terang)	2	0	13	1.0	0.133	23,5%	2 FPS

-----Halaman ini sengaja dikosongkan-----

BAB V
PENUTUP

5.1 Kesimpulan

Berdasarkan percobaan yang telah dilakukan pada pelaksanaan tugas akhir dengan judul *Vision System untuk Identifikasi Shagai pada ABU Robocon 2019*, didapat beberapa kesimpulan sebagai berikut:

a. Sistem yang berhasil dibuat dapat mendeteksi *Shagai* dengan sangat baik yaitu dengan nilai akurasi F-Score sebesar 98,5% untuk *Shagai* Merah dan 98,65% untuk *Shagai* Biru.
b. Sistem yang berhasil dibuat dapat mengidentifikasi poin *Shagai* dengan sangat baik yaitu dengan akurasi sebesar 91,15% untuk *Shagai* Merah dan 90,8% untuk *Shagai* Biru.
c. Sistem yang berhasil dibuat dapat mendeteksi *Shagai* dengan optimal yaitu pada pencahayaan sebesar 18 lux.
d. Sistem yang berhasil dibuat memiliki kecepatan pemrosesan citra rata-rata sebesar 2 FPS.

5.2 Saran

Dari hasil pengujian *vision system* untuk identifikasi *Shagai* pada ABU Robocon 2019, terdapat beberapa hal yang dapat diperbaiki lagi. Untuk hasil deteksi agar lebih baik, bisa ditambahkan lagi dataset gambar yang lebih banyak dan lebih bervariasi baik dari jarak maupun hadap sisi, sehingga model bisa mengenali objek dengan baik. Untuk algoritma penentuan poin, kedepannya bisa ditambahkan beberapa logika untuk bisa mengetahui kemiringan objek.

-----Halaman ini sengaja dikosongkan-----

DAFTAR PUSTAKA

[1] ABU Asia-Pacific Robot Contest 2019 Ulaanbaatar, Mongolia Organizing Committee, *Robocon 2019 Mongolia Rule*. 2018.

[2] ABU Asia-Pacific Robot Contest 2019 Ulaanbaatar, Mongolia Organizing Committee, *Shagai Scoring*. 2018.

[3] ABU Asia-Pacific Robot Contest 2019 Ulaanbaatar, Mongolia Organizing Committee, *Shagai*. 2018.

[4] D. S. Y. Kartika dan D. Herumurti, "Koi fish classification based on HSV color space," dalam *2016 International Conference on Information & Communication Technology and Systems (ICTS)*, Surabaya, Indonesia, 2016, hlm. 96–100.

[5] C.-H. Su, H.-S. Chiu, dan T.-M. Hsieh, "An efficient image retrieval based on HSV color space," dalam *2011 International Conference on Electrical and Control Engineering*, Yichang, China, 2011, hlm. 5746–5749.

[6] P. S. Tantra, "PENENTUAN POSISI ROBOT SEPAK BOLA BERODA BERDASARKAN PENGINDRAAN VISUAL," hlm. 88, 2018.

[7] L. Feng, L. Xiaoyu, dan C. Yi, "An efficient detection method for rare colored capsule based on RGB and HSV color space," dalam *2014 IEEE International Conference on Granular Computing (GrC)*, Noboribetsu, Japan, 2014, hlm. 175–178.

[8] "TensorFlow," *TensorFlow*. [Daring]. Tersedia pada: https://www.tensorflow.org/. [Diakses: 23-Mei-2019].

[9] N. Buduma dan N. Locascio, "Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms," hlm. 298.

[10] M. A. Ponti, L. S. F. Ribeiro, T. S. Nazare, T. Bui, dan J. Collomosse, "Everything You Wanted to Know about Deep Learning for Computer Vision but Were Afraid to Ask," dalam *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T)*, Niterói, 2017, hlm. 17–41.

[11] L. Yu, X. Chen, dan S. Zhou, "Research of Image Main Objects Detection Algorithm Based on Deep Learning," dalam *2018 IEEE 3rd International Conference on Image, Vision and Computing (ICIVC)*, Chongqing, 2018, hlm. 70–75.

[12] W. Liu *dkk.*, "SSD: Single Shot MultiBox Detector," *ArXiv151202325 Cs*, vol. 9905, hlm. 21–37, 2016.

[13] L. Ranalli, L. Di Stefano, E. Plebani, M. Falchetto, D. Pau, dan V. D'Alto, "Automated Generation of a Single Shot Detector C Library from High Level Deep Learning Frameworks," dalam *2018 IEEE 4th International Forum on Research and Technology for Society and Industry (RTSI)*, Palermo, 2018, hlm. 1–4.
[14] Chengcheng Ning, Huajun Zhou, Yan Song, dan Jinhui Tang, "Inception Single Shot MultiBox Detector for object detection," dalam *2017 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, Hong Kong, Hong Kong, 2017, hlm. 549–554.
[15] C. Szegedy *dkk.*, "Going Deeper with Convolutions," *ArXiv14094842 Cs*, Sep 2014.
[16] "About." [Daring]. Tersedia pada: https://opencv.org/about/. [Diakses: 23-Mei-2019].
[17] "Google Colaboratory." [Daring]. Tersedia pada: https://colab.research.google.com/notebooks/welcome.ipynb. [Diakses: 23-Mei-2019].
[18] "HD Webcam C270 - Logitech Support." [Daring]. Tersedia pada: https://support.logitech.com/en_us/product/hd-webcam-c270. [Diakses: 23-Mei-2019].

LAMPIRAN

A. Program Pengambilan Dataset

```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <stdio.h>

using namespace cv;
using namespace std;

int ret, i = 0, c = 0;
char buffer[100];

int main() {
	Mat src;

	VideoCapture capture;

	capture.open(0);

	while (1)
	{
		capture.read(src);

		namedWindow("Ori",WINDOW_AUTOSIZE);
		imshow("Ori", src);

		c++;

		if (c > 10)
		{
			sprintf_s(buffer,
"dataset/m40/m40_%d.jpg", i);
			i++;

			cout << i << endl;
```

```
                imwrite(buffer, src);

                c = 0;
            }

            if (waitKey(1) == 27) break;

        }

        return 0;
}
```

B. Program Deteksi dan Penentuan Poin *Shagai*

```
#include <opencv2/opencv.hpp>
#include <opencv2/dnn.hpp>
#include <iostream>

using namespace cv;
using namespace std;
using namespace dnn;

void colorDetect(Mat src, Mat* dst, int color, int* status, Point2f*
center)
{
        int H_MIN = 0;
        int H_MAX = 179;
        int S_MIN = 0;
        int S_MAX = 255;
        int V_MIN = 0;
        int V_MAX = 255;

        if (color == 1)
        {
            H_MIN = 88;
            H_MAX = 179;
            S_MIN = 98;
            S_MAX = 255;
```

```
                V_MIN = 68;
                V_MAX = 255;
        }
        else if (color == 2)
        {
                H_MIN = 9;
                H_MAX = 34;
                S_MIN = 100;
                S_MAX = 255;
                V_MIN = 80;
                V_MAX = 255;
        }
        else if (color == 3)
        {
                H_MIN = 0;
                H_MAX = 179;
                S_MIN = 8;
                S_MAX = 91;
                V_MIN = 124;
                V_MAX = 255;
        }
        else if (color == 4)
        {
                H_MIN = 142;
                H_MAX = 179;
                S_MIN = 0;
                S_MAX = 255;
                V_MIN = 108;
                V_MAX = 255;
        }

        vector<vector<Point> > contours;
        vector<Vec4i> hierarchy;

        cvtColor(src, *dst, COLOR_BGR2HSV);
        inRange(*dst, Scalar(H_MIN, S_MIN, V_MIN),
Scalar(H_MAX, S_MAX, V_MAX), *dst);
        findContours(*dst, contours, hierarchy,
RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
```

```
	int biggest = 0, max = 0;
	int area;
	int detect = 0;

	for (unsigned int i = 0; i < contours.size(); i++)
	{
		area = contourArea(contours[i]);
		if (area > 500)
		{
			detect++;
			if (area > max)
			{
				max = area;
				biggest = i;
			}
			else
			{
				max = max;
			}
		}
	}

	if (detect != 0)
	{
		vector<Moments> mu(contours.size());
		Point c;
		mu[biggest] = moments(contours[biggest]);
		c.x = (mu[biggest].m10 / (mu[biggest].m00 +
1e-5));
		c.y = (mu[biggest].m01 / (mu[biggest].m00 +
1e-5));

		*center = Point(c.x, c.y);
		*status = color;
	}
	else
		*status = 0;
}
```

```
int main() {

	int mode = 0; // 0 untuk shagai merah, 1 untuk shagai
biru
	Mat frame;
	Mat frame1;
	Mat hsv;

	Mat Side, golden, silver;
	Point2f centerSide, centerGolden, centerSilver;

	Mat blob;
	Mat outs;
	Net net;

	net =
readNetFromTensorflow("frozen_inference_graph.pb",
"graph.pbtxt");

	int i = 0;
	char buffer[100];
	int label = 0;

	VideoCapture cap;
	cap.open(0);

	while (1)
	{
		cap.read(frame);

		frame1 = frame.clone();

		net.setInput(blobFromImage(frame, 1.0,
Size(300, 300), true, false));
		net.forward(outs);

		Mat detectionMat(outs.size[2], outs.size[3],
CV_32F, outs.ptr<float>());

		int total_poin = 0;
```

```
                int poin;

                float confidenceThreshold = 0.6;
                for (int i = 0; i < detectionMat.rows; i++)
                {
                        float confidence =
detectionMat.at<float>(i, 2);

                        if (confidence > confidenceThreshold)
                        {
                                int idx =
static_cast<int>(detectionMat.at<float>(i, 1));
                                int xLeftBottom =
static_cast<int>(detectionMat.at<float>(i, 3) * frame.cols);
                                int yLeftBottom =
static_cast<int>(detectionMat.at<float>(i, 4) * frame.rows);
                                int xRightTop =
static_cast<int>(detectionMat.at<float>(i, 5) * frame.cols);
                                int yRightTop =
static_cast<int>(detectionMat.at<float>(i, 6) * frame.rows);

                                if (xLeftBottom < 0)
                                        xLeftBottom = 0;
                                if (yLeftBottom < 0)
                                        yLeftBottom = 0;
                                if (xRightTop > 640)
                                        xRightTop = 640;
                                if (yRightTop > 480)
                                        yRightTop = 480;

                                Rect object((int)xLeftBottom,
(int)yLeftBottom,
                                        (int)(xRightTop -
xLeftBottom),
                                        (int)(yRightTop -
yLeftBottom));

                                Mat roi = frame(object);

                                int status = 0;
```

```
                                        if (mode == 0)
colorDetect(roi, &Side, 4, &status, &centerSide);
                                        else if (mode == 1)
colorDetect(roi, &Side, 1, &status, &centerSide);

                                        if (status == 1 || status == 4)
                                        {
                                                imshow("Side",
Side);

                                                colorDetect(roi,
&golden, 2, &status, &centerGolden);
                                                if (status == 2)
                                                {

        imshow("golden", golden);

                                                        float
derajat;

                                                        derajat =
atan2((centerSide.y - centerGolden.y), (centerSide.x -
centerGolden.x)) * 57.296;
                                                        cout <<
"derajat 1 : " << derajat << endl;

                                                        if (derajat >
60 && derajat < 125)
                                                        {
                                                                if
((centerSide.y - centerGolden.y) > abs(centerSide.x -
centerGolden.x))
                                                                {

        label = 2;
                                                                }

        else
                                                                {
```

```
        label = 1;
                                                        }
                                                }
                                                else

        label = 1;
                                        }
                                        else
                                        {
                                                status = 5;
                                        }

                                        if (status == 5)
                                        {

        colorDetect(roi, &silver, 3, &status, &centerSilver);

                                                if (status ==
3)
                                                {

        imshow("silver", silver);


        float derajat;


        derajat = atan2((centerSide.y - centerSilver.y),
(centerSide.x - centerSilver.x)) * 57.296;

        cout << "derajat 2 : " << derajat << endl;

                                                        if
(derajat > 60 && derajat < 125)
                                                        {

        if ((centerSide.y - centerSilver.y) > abs(centerSide.x -
centerSilver.x))
```

```
        {
                label = 3;
        }
        else
        {
                label = 1;
        }                                                       }
        else
        label = 1;
                                                        }
                                                        else
        label = 1;
                                                }
                                        }
                                        imshow("roi", roi);
                                        rectangle(frame1,
Rect(object.x, object.y - 25, 45, 25), Scalar(0, 255, 70), -1);
                                        rectangle(frame1, object,
Scalar(0, 255, 70), 8);
                                        if (label == 1)
                                        {
                                                putText(frame1,
format("20"), Point(object.x + 5, object.y - 5),
FONT_HERSHEY_SIMPLEX, 0.7, Scalar(0, 0, 0), 2);
```

```
                                        poin = 20;
                                    }
                                    else if (label == 2)
                                    {
                                        putText(frame1,
format("50"), Point(object.x + 5, object.y - 5),
FONT_HERSHEY_SIMPLEX, 0.7, Scalar(0, 0, 0), 2);
                                        poin = 50;
                                    }
                                    else if (label == 3)
                                    {
                                        putText(frame1,
format("40"), Point(object.x + 5, object.y - 5),
FONT_HERSHEY_SIMPLEX, 0.7, Scalar(0, 0, 0), 2);
                                        poin = 40;
                                    }
                                    //imshow("red", red);

                                    total_poin += poin;
                                }
                            }

                            putText(frame1, format("Total Poin = %d",
total_poin), Point(30, 30), FONT_HERSHEY_SIMPLEX, 0.9,
Scalar(0, 0, 0), 2);

                            imshow("output", frame1);

                            waitKey(0);
                            destroyAllWindows();
                        }
}
```

BIODATA PENULIS

Fajar Luhung Parasdyo lahir di Jepara pada tanggal 30 Agustus 1997. Penulis memulai kehidupan perkuliahan pada tahun 2015 di Departemen Teknik Elektro, Fakultas Teknologi Elektro, Institut Teknologi Sepuluh Nopember Surabaya. Selama masa perkuliahan, penulis aktif dalam berbagai kegiatan kepanitiaan dan organisasi khususnya di bidang robotika. Selain itu, penulis juga menjadi asisten praktikum di bidang studi elektronika. Penulis bisa dihubungi melalui e-mail: fajarluhung@gmail.com.