TUGAS AKHIR - EE 184801

Perancangan Sistem Pump-Probe Spectroscopy Sebagai Metode Dalam Analisis Dinamika Elektron pada Larutan

Reynaldi Gilang Mulyawan
NRP 07111640000108

Dosen Pembimbing
Dr. Muhammad Rivai, S.T., M.T.
Dr. Isnaeni, M.Sc.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro Dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

Halaman ini sengaja dikosongkan

TUGAS AKHIR - EE 184801

Perancangan Sistem *Pump-Probe Spectroscopy* Sebagai Metode Dalam Analisis Dinamika Elektron pada Larutan

Reynaldi Gilang Mulyawan
NRP 07111640000108

Dosen Pembimbing
Dr. Muhammad Rivai, S.T., M.T.
Dr. Isnaeni, M.Sc

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro Dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

Halaman ini sengaja dikosongkan

FINAL PROJECT - EE 184801

Design of Pump-Probe Spectroscopy System as A Method of Analysis of Electronic Dynamic of Solutions

Reynaldi Gilang Mulyawan
NRP 07111640000108

Supervisor
Dr. Muhammad Rivai, S.T., M.T.
Dr. Isnaeni, M.Sc

ELECTRICAL ENGINEERING DEPARTMENT
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2020

Halaman ini sengaja dikosongkan

PERNYATAAN KEASLIAN
TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul :

"Perancangan Sistem *Pump-Probe Spectroscopy* Sebagai Metode Dalam Analisis Dinamika Elektron pada Larutan"

adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diizinkan 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, 24 Juni 2020

Reynaldi Gilang Mulyawan
NRP. 07111640000108

Halaman ini sengaja dikosongkan

Perancangan Sistem *Pump-Probe Spectroscopy* Sebagai Metode Dalam Analisis Dinamika Elektron pada Larutan

TUGAS AKHIR

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

Menyetujui :
Dosen Pembimbing I

Dr. Muhammad Rivai, ST, MT.
NIP. 131647788

SURABAYA
JUNI, 2020

Halaman ini sengaja dikosongkan

Perancangan Sistem *Pump-Probe Spectroscopy* Sebagai Metode Dalam Analisis Dinamika Elektron pada Larutan

TUGAS AKHIR

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

Menyetujui :
Dosen Pembimbing II

Dr. Isnaeni, M.Sc.
NIP. 132301043

SURABAYA
JUNI, 2020

Halaman ini sengaja dikosongkan.

Perancangan Sistem *Pump-Probe Spectroscopy* Sebagai Metode Dalam Analisis Dinamika Elektron pada Larutan

Reynaldi Gilang Mulyawan
07111640000108

Dosen Pembimbing : 1. Dr. Muhammad Rivai, S.T., M.T.
2. Dr. Isnaeni, M.Sc.

ABSTRAK

Spektrokmetri adalah studi dalam bidang optoelektronika yang mempelajari tentang relasi antara suatu zat dengan radiasi elektromagnetik. Namun, berbagai metode spektroskopi seperti UV-Vis maupun Raman Spectroscopy tidak mampu untuk mendeskripsikan analisis bahan pada skala dinamika elektron, yang merupakan hal yang esensial dalam menganalisis bahan semikonduktor, seperti carbon quantum dot. Dalam tugas akhir ini akan dibuat rancang bangun sistem pump-probe spectroscopy yang dapat memodelkan dinamika struktur elektron pada suatu zat dalam skala waktu femtosekon (10^{-15} second). sekon). Sistem akan dirancang dalam bentuk simulasi berbasis Lindbladian Master Equation dan setup eksperimen. Sistem menggunakan cahaya laser dengan panjang gelombang 1064 nm yang akan dikenakan pada zat uji. Laser dipisahkan menjadi dua beam menggunakan sebuah beam splitter, yakni pump dan probe. Cahaya laser pump akan digunakan untuk mengeksitasi zat yang diujikan, sedang cahaya laser probe digunakan untuk melakukan probing pada zat tersebut untuk ditangkap oleh photodetector. Pengaturan delay stage pada sistem akan membantu dalam menghasilkan data struktur elektorn yang berubah terhadap waktu. Kemudian data diklasifikasi dengan recurrent neural network untuk memprediksi jenis bahan uji. Terdapat hasil berupa nilai absorbansi setinggi 0.3 unit absorbansi baik untuk sample berupa larutan carbon quantum dot maupun quantum dot, dan dari pengklasifikasi data didapatkan akurasi 90% untuk klasifikasi data pada sample larutan quantum dot dan 81% untuk larutan carbon quantum dot. Hal tersebut digunakan untuk membuktikan bahwa hasil dari komputasi simulator sama dengan hasil setup eksperimen, serta pengklasifikasi yang telah dibuat dapat mencapai akurasi yang tinggi.

Kata Kunci : *Lindbladian Master Equation*, Optoelektronika, *Recurrent Neural Network*, Spektroskopi

Halaman ini sengaja dikosongkan

Design of Pump-Probe Spectroscopy System as A Method of Analysis of Electronic Dynamic of Solutions

Reynaldi Gilang Mulyawan
07111640000108

Supervisor : 1. Dr. Muhammad Rivai, ST, MT.
2. Dr. Isnaeni, M.Sc.

ABSTRACT

Spectrometry is a study in the field of optoelectronics that studies the relationship between a substance with electromagnetic radiation. However, various spectroscopic methods such as UV-Vis or Raman Spectroscopy are unable to describe material analysis at the electron dynamics scale, which is essential in analyzing semiconductor materials, such as carbon quantum dots. In this final project will be designed a pump-probe spectroscopy system that can model the dynamics of the electron structure of a substance on the femtosecond time scale (10^{-15} second). The system will be designed in the form of a Lindbladian Master Equation based simulation and an experimental setup. The system uses a laser light with a wavelength of 1064 nm which will be subject to the test substance. The laser is separated into two beams using a beam splitter, the pump and the probe. The laser pump light will be used to excite the substance being tested, while the laser light probe is used to probing the substance to be captured by the photodetector. Setting the delay stage on the system will help in generating data on the electronic structure that changes with time. Then the data is classified with recurrent neural network to predict the type of test material. There are results in the form of absorbance values as high as 0.3 units of absorbance both for the sample in the form of carbon quantum dot and quantum dot solutions, and from the classification of the data obtained 90% accuracy for the classification of data in the sample quantum dot solution and 81% for carbon quantum dot solutions. It is used to prove that the results of the computational simulator experience the same phenomenon as the results obtained in the experimental setup, and the classifiers that have been made can achieve high accuracy

Keyword : Linbladian Master Equation, Optoelectronics, Recurrent Neural Network, Spectroscopy,

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan YME karena atas rahmat dan penyertaan-Nya, penulis dapat mengerjakan dan menuntaskan laporan tugas akhir ini dengan judul "Perancangan Sistem *Pump-Probe Spectroscopy* Sebagai Metode Dalam Analisis Dinamika Elektron pada Larutan". Tujuan dari pembuatan laporan tugas akhir ini tidak lain sebagai salah satu prasyarat kelulusan penyelesaian program pendidikan sarjana dan mata kuliah yang wajib ditempuh di Departemen Teknik Elektro Institut Teknologi Sepuluh Nopember.

Selama proses pengerjaan penulis menyadari sebagai manusia biasa tanpa bantuan, dukungan dan arahan dari banyak pihak tanpanya mungkin saja laporan tugas akhir ini tidak bisa diselesaikan terlebih ditengah pandemi COVID-19, oleh karenanya secara khusus penulis ingin mengucapkan terima kasih kepada:

1. Bapak Dr. Muhammad Rivai, S.T., M.T. selaku dosen pembimbing pertama dan bapak Dr. Isnaeni, M.Sc dari Pusat Penelitian Fisika – Lembaga Ilmu Pengetahuan Indonesia selaku dosen pembimbing kedua yang banyak memberikan bantuan baik gagasan topik tugas akhir serta bimbingan kepada penulis selama mengerjakan tugas akhir.
2. Bapak Ronny Mardianto, ST, MT, PhD., Bapak Ir. Tasripan, MT., Bapak Muhammad Attamimi, B.Eng., M.Eng., Ph.D., serta Bapak Haris Pirngadi, ST, MT, selaku dosen penguji yang telah memberikan arahan kepada penulis
3. Bapak Dedet Chandra Riawan, S.T., M.Eng, Ph.D. selaku Kepala Departemen Teknik Elektro ITS.
4. Seluruh dosen dan staf pegawai Departemen Teknik Elektro ITS.
5. Segenap peneliti dan staf pegawai Pusat Penelitian Fisika – Lembaga Ilmu Pengetahuan Indonesia.
6. Orang-orang terdekat, khususnya Laksmi Paraskevi Nugrahadi yang telah membesarkan penulis serta dukungan moral, materiil, dan ilmu yang diterapkan dalam pengerjaan tugas akhir.

Terlepas dari itu semua, penulis menyadari masih banyak kekurangan baik dari segi penyajian laporan dan materi oleh karenanya penulis menerima kritik dan saran yang diberikan demi pengembangan

alat yang lebih baik. Akhir kata, semoga dengan diadakannya penelitian tugas akhir ini dapat memberikan manfaat bagi banyak pihak.

Surabaya, 13 Agustus 2020

Reynaldi Gilang Mulyawan

Halaman ini sengaja dikosongkan

DAFTAR ISI

Halaman ini sengaja dikosongkan.

DAFTAR GAMBAR

BAB 1
PENDAHULUAN

Pada bab 1 dijelaskan mengenai latar belakang dari penelitian yang telah dilakukan. Berawal dari permasalahan sistem yang terjadi, peneliti berkeinginan untuk mengembangkan sistem melalui algoritma-algoritma yang ada untuk dilakukan pengujian dari beberapa model. Pada penelitian ini diawali dengan melakukan studi literatur, perancangan sistem, pengujian dan analisis. Dalam bab ini akan dijelaskan mengenai latar belakang, perumusan masalah, batasan masalah, tujuan, sistematika, serta metodologi yang digunakan dalam penelitian.

1.1 Latar Belakang

Spektroskopi dapat digunakan untuk melakukan pengukuran subatomik [1], yang mempelajari tentang interaksi antara radiasi elektromagnetik dengan zat uji. Pada umumnya, spektroskopi kerap menggunakan cahaya, khususnya yang berasal dari laser, sebagai sumber radiasi elektromagnetik [2].

Pada skala pikosekon, laser dapat digunakan untuk mengukur absorbansi maupun emisi cahaya suatu bahan [3] Hal tersebut masih belum dapat menjelaskan lebih lanjut tentang sifat suatu material, seperti bagaimana sifat optis dari bahan tersebut dapat dipelajari lebih lanjut daripada hanya sekadar sifat absorbansi maupun emisinya., serta perihal lain seperti kemurnian bahan semikonduktor dan kestabilan zat kimia. Guna mengatasi hal tersebut, digunakanlah pulsa laser dalam waktu skala femtosekon (10^{-15} sekon), sehingga laser ini dapat digunakan untuk memodelkan dinamika elektron dalam suatu atom.

Metode *Pump-Probe* ini merupakan metode yang paling sederhana dari semua permodelan dinamika elektron yang ada karena metode ini hanya menggunakan satu sumber laser yang berkas cahayanya dibagi menjadi dua bagian untuk eksitasi (*pump*) dan pengukuran (*probe*) dengan menggunakan *beam splitter*. Spektroskopi *Pump-Probe* diharapkan mampu untuk mempelajari zat dalam skala dinamika elektron yang dapat memiliki dampak besar terhadap analisis bahan semikonduktor dalam segi kemurnian, kestabilan zat kimia setelah

melalui reaksi tertentu, serta mengembangkan akuisisi data yang baik dan benar dalam analisis zat yang diujikan.

1.2 Perumusan Masalah

Berikut ialah perumusan masalah yang ada dalam pengambilan topik ini:

1) Bagaimana sistem *pump-probe spectroscopy* dapat digunakan untuk menganalisis *ultrafast electron dynamics* tiap zat uji secara baik dan benar?
2) Bagaimana teknik akuisisi data yang benar secara definisi dalam penggunaan sistem pump-probe ini dalam melakukan analisis zat uji, serta memperluas metode akuisisi data tersebut?

1.3 Batasan Masalah

Dalam penelitian ini permasalahan yang dibahas memiliki batasan masalah sebagai berikut :

1. Data yang digunakan untuk training dan testing secara mandiri melalui percobaan eksperimental di PPF LIPI.
2. Dalam implementasi pemrograman simulator dan klasifikasi data sample menggunakan pemrograman berbasis python 3.

1.4 Tujuan Penelitian

Tujuan yang dirumuskan dalam penyusunan topik tugas akhir ini adalah sebagai berikut:

1) Mampu membangun sistem *pump-probe* yang dapat menganalisis *ultrafast electron dynamics* menggunakan laser dengan pulsa pada skala waktu femtosekon.
2) Mampu mengembangkan teknik akuisisi data yang definitif untuk setiap bahan uji yang digunakan dalam sistem *pump-probe spectroscopy* dengan metode *neural network*.

.

1.5 Metodologi

1.5.1 Studi Literatur Sistem Pump Probe Spectroscopy

Pada studi literatur dilakukan pengumpulan berbagai literatur atau sumber sebagai rujukan yang berupa teori, data, ataupun penelitian yang dapat menunjang dalam penulisan tugas akhir ini. Literatur ini dapat diambil dari buku-buku, jurnal-jurnal, artikel, dan forum diskusi.

1.5.2 Perancangan Perangkat Keras (*hardware*) sistem Pump-Probe Spectroscopy dan Akuisisi Data Sistem

Pada tahap ini dilakukan perancangan perangkat keras berupa penyusunan *setup* laser pulsa skala femtosekon. Adapun tahap ini juga meliputi *trouble shooting* bila terjadi kesalahan desain.

1.5.3 Perancangan Perangkat Lunak Berupa Simulator dan Peng-klasifikasi

Pada tahap ini dilakukan perancangan perangkat lunak berupa simulator sistem *pump-probe spectroscopy* beserta peng-klasifikasi menggunakan *recurrent neural network*.

1.5.4 Tahap Pengujian Sistem Pump-Probe Spectroscopy

Pada tahap pengujian, alat diuji di berbagai sample dan kondisi yang berbeda untuk mengetahui karakteristik dan kekurangan dari alat tersebut sehingga dapat dilakukan perbaikan untuk menyesuaikan. Adapun sample yang akan digunakan ialah larutan *carbon dot* dari jahe dan lengkuas, serta data yang dihasilkan oleh simulator.

1.5.5 Analisis dan Evaluasi Data

Data yang telah diakuisisi melalui metode *pump-probe* pada program NI LabVIEW 2018 pada setup eksperimen dan yang dihasilkan oleh simulator kemudian akan disimpan dalam bentuk file *Microsoft Excel*. Data yang dihasilkan kemudian akan digunakan untuk digunakan sebagai bahan *training* pada *recurrent neural network* yang telah didesain, sehingga program tersebut dapat digunakan untuk memprediksi akurasi dari prediksi *classifier*/pengklasifikasi.

1.5.6 Penulisan Buku Tugas Akhir & Jurnal POMITS

Pada tahap ini dilakukan penyusunan dan penulisan laporan tugas akhir sesuai dengan struktur yang telah ditentukan. Terdapat tiga bagian besar untuk dimasukkan ke dalam laporan tugas akhir, yaitu bagian awal yang memuat bahan-bahan preliminer, bagian inti/pokok yang memuat naskah utama dari tugas akhir, dan bagian akhir yang memuat bahan-bahan referensi. Penyusunan dan penulisan laporan tugas akhir dilakukan bersamaan dengan perancangan dan pengujian alat.

1.6 Sistematika Penulisan

Pada penelitian ini disusun sistematika penulisan sesuai dengan penjelasan 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 DASAR TEORI

Dalam bab ini akan dijelaskan terkait teori dasar dari penelitian yang dilakukan. Teori dasar yang mendasari konsep dari sistem *pump-probe spectroscopy*, akuisisi serta klasifikasi data sample masing-masing dengan menggunakan software NI-LabVIEW 2018 dan metode recurrent neural network serta library yang digunakan untuk mendukung proses sistem.

BAB 3 PERANCANGAN SISTEM

Pada bab ini dilakukan perancangan sistem untuk klasifikasi citra yang diawali dengan akuisisi data sample melalui setup eksperimental pada laboratorium, dan dilanjutkan dengan perancangan sistem simulator dengan data eksperimental sebagai bukti penunjang dan recurrent neural network sebagai metode klasifikasi data

BAB 4 PENGUJIAN DAN ANALISIS

Dalam pengujian dan analisis data akan dilakukan proses implementasi untuk akuisisi data dari sistem pump-probe spectroscopy dari laboratorium, dan data tersebut masing-masing akan digunakan untuk verifikasi konsep sistem ini dan juga sebagai bahan training pada recurrent neural network.

BAB 5 KESIMPULAN

Dalam bab ini akan dijelaskan suatu kesimpulan terkait hasil yang telah dicapai pada penelitian yang telah dilakukan

Halaman ini sengaja dikosongkan

BAB 2
TINJAUAN PUSTAKA

Dalam Bab ini dijelaskan mengenai teori-teori dasar untuk membangun algoritma *recurrent neural network* khususnya dalam klasifikasi citra. Arsitektur dan teori-teori pendukung lainnya dijelaskan juga dengan tujuan untuk memberikan gambaran umum mengenai deteksi objek pada citra.

2.1 Cahaya dan Optika non-linier

Cahaya merupakan gelombang elektromagnetik yang merambat dalam ruang tiga dimensi, baik dalam vakum maupun dalam suatu zat serta berubah terhadap waktu. Cahaya dapat dideskripsikan melalui persamaan Maxwell bentuk differensial yang mendefinisikan sifat medan listrik dan magnet dalam media tanpa muatan ataupun arus [2] diberikan sebagai berikut:

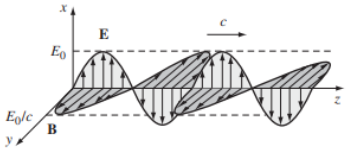
$$\nabla . \mathbf{E} = 0 \tag{2.1}$$

$$\nabla . \mathbf{B} = 0 \tag{2.2}$$

$$\nabla \text{ x } \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{2.3}$$

$$\nabla \text{ x } \mathbf{B} = \mu_0 \varepsilon_0 \frac{\partial \mathbf{E}}{\partial t} \tag{2.4}$$

Di mana **E** dan **B** masing-masing ialah medan listrik dan magnet. Persamaan (1) dan (2) adalah Hukum Gauss mengenai medan listrik dan magnet, persamaan (3) adalah Hukum Faraday dan persamaan (4) ialah Hukum Ampere.



Gambar 2.1 Propagasi gelombang eletromagnetik dalam ruang tiga dimensi [2].

Terlihat dari gambar 2.1 bahwa kuat medan magnet **B** selalu lebih lemah satu faktor c, yakni kecepatan cahaya, dibandingkan medan listrik. Dari sini, dapat dilihat bahwa dalam propagasi cahaya, kuat medan magnet dalam gelombang cahaya dapat diabaikan. Agar dapat mendefinisikan cahaya sebagai bentuk gelombang elektromagnetik, persamaan (3) dapat diturunkan sehingga:

$$\nabla\, x\, (\nabla\, x\, \mathbf{E}) = \nabla.\,(\nabla\,.\,\mathbf{E}) - \nabla^2\mathbf{E} = \nabla\,\mathrm{x}\left(-\frac{\partial \mathbf{B}}{\partial t}\right)$$

$$= -\frac{\partial}{\partial t}(\nabla\,\mathrm{x}\,\mathbf{B}) = -\mu_0\varepsilon_0\frac{\partial^2\mathbf{E}}{\partial t^2} \tag{2.5}$$

Merujuk pada persamaan (2.1), persamaan (2.5) dapat ditulis sebagai:

$$\nabla^2\mathbf{E} = \mu_0\varepsilon_0\frac{\partial^2\mathbf{E}}{\partial t^2} \tag{2.6}$$

Persamaan (2.6) merupakan persamaan differensial parsial yang mendeskripsikan arah rambat gelombang, yang memiliki solusi berupa:

$$\mathbf{E}(\mathbf{r},t) = E(t)\exp(i(\mathbf{k}.\,\mathbf{r} - \omega t + \varphi)) \tag{2.7}$$

Di mana **k** merupakan vector gelombang, ω adalah frekuensi gelombang, dan φ adalah fasa [1]. Medan listrik **E(r,*t*)** menjadi dasar dari derajat polarisasi material, yang didefinisikan dengan persamaan berikut:

$$\mathbf{P}^{(3)}(t) = \chi^{(3)}(t_3,t_2,t_1)\mathbf{E}_3\mathbf{E}_2\mathbf{E}_1 \tag{2.8}$$

Dimana χ merupakan *susceptibility* dari bahan tersebut pada derajat ke-3. Hubungan antara **E** dan **P** sekarang dapat didefinisikan melalui persamaan gelombang Maxwell:

$$\mathbf{E}_s(t) \propto i\mathbf{P}^{(3)} \tag{2.9}$$

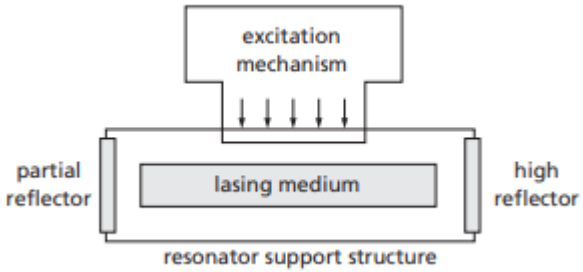
Intensitas dari berkas laser kemudian dapat didefenisikan sebagai kuadrat dari polarisasi dan suspektibilitas pada *homodyne* detection:

$$I_s = \frac{nc}{8\pi}|\mathbf{E}_s(t)|^2 \propto |\mathbf{P}^{(3)}(t)|^2 \propto |\chi^{(3)}|^2 \quad (2.10)$$

Di mana *n* merupakan indeks bias material dan *c* adalah kecepatan cahaya.

2.2 Laser

Laser yang merupakan singkatan dari *Light Amplification by Stimulated Emission and Radiation* adalah berkas cahaya yang bersifat monokromatik, koheren, dan searah/unidireksional [3].



Gambar 2.2 Mekanisme terbangkitnya sinar laser [3].

Emisi energi dalam sebuah atom dapat dibagi menjadi dua bagian, yakni emisi spontan dan emisi terstimulas seperti pada gambar 2.2. Emisi spontan terjadi ketika elektron pada suatu atom turun dari tingkat energi yang lebih tinggi menuju tingkat energi dasar. Perpindahan tersebut melepaskan sejumlah energi dari atom tersebut, sedangkan emisi terstimulasi terjadi ketika suatu atom dikenakan radiasi eksternal yang menyebabkan perpindahan elektron dari tingkat energi dasar menuju tingkat energi selanjutnya.

Adapun elektron dari dalam medium gain ini akan mengalami emisi dari hasil stimulasi luar, sehingga terjadi perpantulan emisi radiasi dari medium di dalam badan resonator. Cahaya laser akan keluar dari

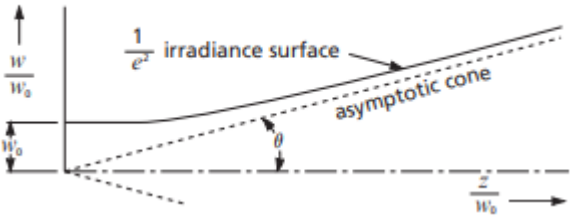
media resonator ketika intensitas yang dihasilkan melebihi dari apa yang pemantul parsial bisa pantulkan.

.

Penyebaran cahaya laser dapat dimodelkan melalui prinsip difraksi cahaya, dimana fenomena tersebut menyebabkan cahaya terpropagasi secara transversal dan susah untuk terkolimasi. Karena hal tersebut, divergensi dari berkas cahaya laser dapat dimodelkan melalui persamaan berikut:

$$z_R = \frac{\pi w_0^2}{\lambda} \tag{2.11}$$

Di mana w_0^2 ialah jari-jari pendaran berkas laser, z_R merupakan jarak *Raleigh*, yang merupakan jarak cahaya laser sebelum terjadi persebaran, serta λ yang merupakan panjang gelombang dari cahaya. Ilustrasi dari pendaran tersebut dapat dilihat pada gambar 2.3.



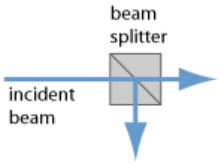
Gambar 2.3 Contoh persebaran Gaussian beam [3].

Sebuah perangkat pembangkit laser terdiri dari beberapa bagian, sebagaimana yang digambarkan pada gambar 2.2. Medium gain sebuah laser, baik dalam bentuk padat, cair, maupun gas akan dieksitasi melalui sebuah mekanisme eksitasi secara terus menerus, contohnya berupa sebuah *flashtube*.

2.3 *Beam Splitter*

Perangkat-perangkat optik yang juga krusial dalam setup *pump-probe spectroscopy* adalah pembelah berkas cahaya). *Beam splitter*

adalah suatu alat optik yang digunakan untuk membelah berkas cahaya menjadi dua atau lebih [4], hal tersebut dapat dilihat pada gambar 2.4.



Gambar 2.4 Beam splitter [4].

Prinsip dari *beam splitter* sendiri berasal dari berasal dari persamaan Fresnel [2], yang menjelaskan tentang sifat cahaya yang dating dari medium satu dikenakan pada suatu medium lainnya dalam sudut tertentu:

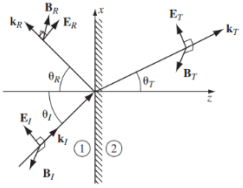
$$E_R = \frac{\alpha-\beta}{\alpha+\beta}E_I \ , E_{0T} = \frac{2}{\alpha+\beta}E_I \tag{2.12}$$

dimana:

$$\alpha = \frac{\cos\theta_T}{\cos\theta_I} \tag{2.13}$$

$$\beta = \frac{\mu_1 n_2}{\mu_2 n_1} \tag{2.14}$$

Dengan $\mu_1$dan μ_2 ialah permeabilitas bahan pada medium 1 dan medium 2, dan n_1 serta n_2 ialah indeks bias dari medium 1 dan 2. Pemantulan cahaya tersebut dapat disimak pada gambar 2.5.



Gambar 2.5 Visualisasi Pemantulan Gelombang Cahaya Melalui Persamaan Fresnel

Agar sebuah beam splitter dapat membagi berkas cahaya laser menjadi dua secara merata, maka persamaan (2.12) harus menjadi:

$$E_R = -\frac{1}{2}E_I\,, E_T = \frac{1}{2}E_I \tag{2.15}$$

2.4 Persamaan Schrödinger dan Lindbladian Master Equation

Persamaan Schrödinger merupakan persamaan diferensial parsial (PDP) yang menjelaskan tentang karakteristik evolusi suatu sistem dari suatu partikel subatomik terhadap waktu [5]:

$$i\hbar\frac{\delta|\varphi>}{\delta t} = H|\varphi> \tag{2.16}$$

Di mana $|\varphi>$ merupakan matriks *ket*, atau matriks yang merepresentasikan keadaan/*state* suatu sistem kuantum. Secara umum, persamaan Schrödinger dapat ditulis ulang sebagai fungsi gelombang, yang merupakan permodelan matematis dari partikel subatomik yang memodelkan distribusi probabilitas dari suatu partikel ditemukan pada keadaan posisi/momentum tertentu seiring berjalannya waktu, serta menggambarkan dualitas partikel-gelombang:

$$i\hbar\frac{\delta\varphi(x,t)}{\delta t} = H\varphi(x,t) \tag{2.17}$$

Di mana H merupakan Hamiltonian sistem yang didefinisikan sebagai total energi kinetik dan potensial yang dimiliki sistem:

$$H = T + V \tag{2.18}$$

Di mana T adalah energi kinetik dan V adalah energi potensial. Persamaan Schrödinger memiliki banyak aplikasi di bidang permodelan zat, yakni dapat digunakan untuk menentukan tingkat energi yang dimiliki oleh sistem serta dinamika absorbsi/emisi suatu zat. Agar dapat merancang sistem spektroskopi yang dapat menunjukkan karakteristik suatu zat dengan gelombang elektromagnetik seperti cahaya, maka sistem tersebut harus memperhitungkan fenomena yang terjadi dalam mekanika kuantum.

Persamaan Hamiltonian dari partikel elektron pada suatu atom ialah sebagai berikut:

$$H = \sum_i^N -\frac{\hbar^2}{2m_e}(\nabla_i^2) - \sum_i^N \frac{Z_e^2}{|r_i|} + \sum_{i<j}^N \frac{e^2}{|r_i - r_j|} \quad (2.19)$$

Dimana *term* pertama ialah energi kinetik dari elektron dengan massa elektron m_e, bagian kedua ialah energi potensial inti atom dengan muatan *Ze*, dan yang ketiga ialah energi potensial antar elektron dengan muatan *e*. Untuk kasus gas Helium yang memiliki dua elektron, persamaan ini menjadi [6]:

$$H = -\frac{\hbar^2}{2m_e}(\nabla_1^2 + \nabla_2^2) - \frac{Z_e^2}{|r_1|} - \frac{Z_e^2}{|r_2|} + \frac{e^2}{|r_1 - r_2|} \quad (2.20)$$

Untuk persamaan Schrödinger pada gas Helium yang dikenakan laser pada sistem *pump-probe*, Hamiltonian sistem ditambahkan dengan medan listrik dari gelombang cahaya laser:

$$H = -\frac{\hbar^2}{2m_e}(\nabla_1^2 + \nabla_2^2) - \frac{Z_e^2}{|r_1|} - \frac{Z_e^2}{|r_2|} + \frac{e^2}{|r_1 - r_2|} + E_0 \cos(\omega t) \quad (2.21)$$

Sehingga hasil akhir dari persamaan (2.17) digunakan untuk memodelkan atom helium yang dikenakan laser menjadi:

$$i\hbar \frac{\delta\varphi(x,t)}{\delta t} = H + E_0 \cos(\omega t))\varphi(x,t) \quad (2.22)$$

Sebagaimana sistem yang terdapat di alam semesta, tidak ada sistem yang murni terisolasi. Hal tersebut tetap berlaku pada mekanika kuantum. Evolusi dari suatu sistem kuantum pun harus mengikuti kaidah-kaidah yang ditetapkan oleh lingkungan sekitarnya.

Metode Lindblad Master Equation merupakan modifikasi dari persamaan Schrödinger yang menggunakan matriks kepadatan/*density matrix* yang merupakan matriks yang mendefinisikan statistika keadaan dari suatu sistem kuantum. *Density matrix* didefinisikan sebagai berikut [7]:

$$\rho = \sum_j p_j |\varphi_j><\varphi_j| \quad (2.23)$$

Di mana $|\varphi_j >< \varphi_j|$ merupakan matriks produk luar/*outer product matrix*. Dari sini, *master equation* Lindblad didefinisikan sebagai:

$$i\hbar\frac{\delta\rho(t)}{\delta t} = [H_{tot}, \rho_{tot}] - \frac{1}{2}(2C_n\rho(t)\mathrm{C_n}^+ - \rho(t)\mathrm{C_n}^+C_n - \mathrm{C_n}^+C_n\rho(t)) \quad (2.24)$$

Di mana ruas kanan terdapat operator komutator:

$$[H_{tot}, \rho_{tot}] = H_{tot}\rho_{tot} - \rho_{tot}H_{tot} \quad (2.25)$$

$$C_n = \sqrt{\gamma_n}A_n \quad (2.26)$$

Di mana C_n adalah *collapse operator* dari atom yang disimulasikan untuk memodelkan disipasi energi dalam sistem kuantum. Lindbladian *master equation* tersedia secara langsung di dalam QuTiP dengan fungsi mesolve() yang akan mempropagasi matriks densitas $\rho(t)$ sebagai persamaan differensial biasa [8]. Yang membuat *master equation* lebih unggul dari persamaan Schrödinger biasa ialah proses yang menggambarkan disipasi dalam sistem kuantum karena interaksinya dengan lingkungan. Interaksi lingkungan ini ditentukan oleh operator di mana sistem berpasangan dengan lingkungan.

Menentukan hasil dari rata-rata probabilitas sistem kuantum dapat dilakukan dengan mudah dengan matriks densitas, yakni dengan menggunakan operasi *trace*, yakni berupa jumlahan dari semua anggota diagonal matriks densitas tersebut:

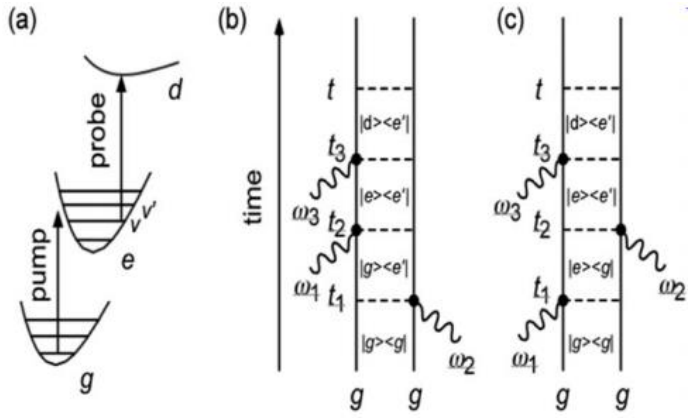
$$< \rho > = Tr(\rho(t) \quad (2.27)$$

2.5 Prinsip Kerja Sistem *pump-probe spectroscopy*

Pump-probe spectroscopy masuk ke dalam ranah optik non-linier yang mempelajari tentang sifat optis dari benda non-linier. Dalam hal tersebut, sifat optis dari sebuah material ditentukan oleh medan magnet cahaya yang datang ke dalam benda tersebut. Sebagai contoh, indeks bias dari suatu material dapat bersifat spesifik terhadap sifat dari cahaya datang [9].

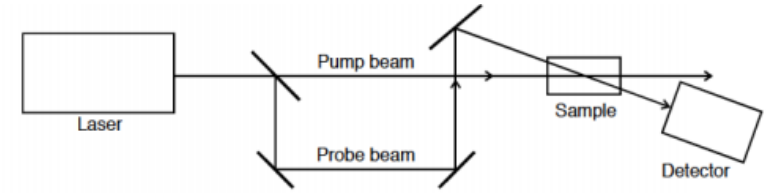
Sistem *pump-probe* sendiri merupakan sistem yang dinamis [10], dikarenakan pulsa laser yang dipakai memiliki nilai delay yang berbeda.

Hal ini memungkinkan pengguna dari sistem ini untuk menganalisis dinamika struktur elektron dalam suatu sample yang beraneka ragam, dari larutan biasa, *quantum dot*, bahan semikonduktor, hingga *alloy* logam. Adapun sistem *pump-probe* ini perlu diperturbasi dari keadaan setimbang, dengan cara mendekatkan sumber laser *pump* kepada sample agar berkas laser tersebut tiba terlebih dahulu daripada probe.



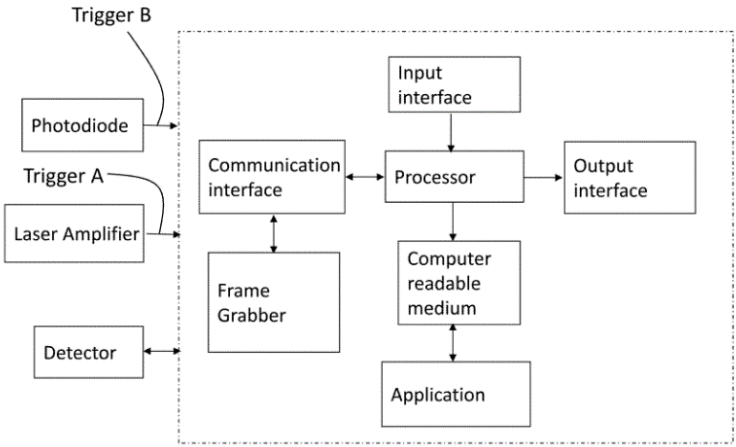
Gambar 2.6 (a) Skema umum pengukuran pump-probe pada tiga tingkat elektronik yang berbeda. (b) menyatakan emisi pada t_1 dan absorpsi pada $t_2$dan t_3, sedangkan (c) menyatakan emisi pada $t_2$dan absorpsi pada t_1 dan t_3 [10].

Seperti yang telah ketahui, atom memiliki struktur elektron yang berbeda-beda, tergantung dari bilangan kuantum atom tersebut. Dengan mengeksitasi elektron dalam atom tersebut, kita dapat mengetahui sifat-sifat yang dimiliki atom tersebut baik unsur individu maupun dalam suatu senyawa. Intensitas dari berkas laser *pump* akan selalu lebih besar daripada intensitas laser *probe*. Berkas laser tersebut memiliki panjang pulsa yang sangat pendek (*ultrashort pulse)* [11].



Gambar 2.7 Skematika Umum Sistem Pump-Probe Spectroscopy [11]

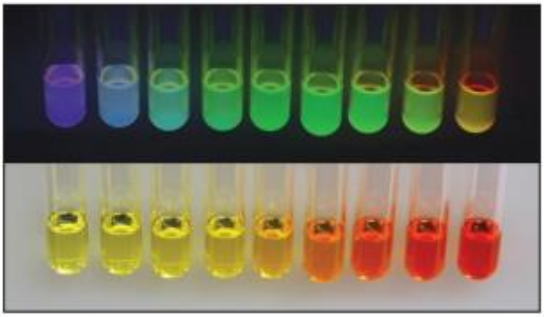
Merujuk pada Gambar 6, sinyal pump yang didefinisikan berdasarkan persamaan (7) bekerja pada tingkat dasar/ground state (g) pada waktu t1 dan t2 untuk membangkitkan *vibrational coherence* (|e, v>,<e, v'|) diantara tingkat v dan v' dalam tingkat yang tereksitasi (e). Adapun sinyal probe yang di-delay pada waktu t_3 juga tiba pada tingkat elektronik selanjutnya (d). Jika *vibrational coherence* masih ada pada saat waktu delay Dt = t_3 - t_2, transimisi dari pulsa probe tersebut dimodulasi sesuai dengan pergerakan partikel yang koheren.



Gambar 2.8 Blok diagram sistem *pump-probe spectroscopy* [12].

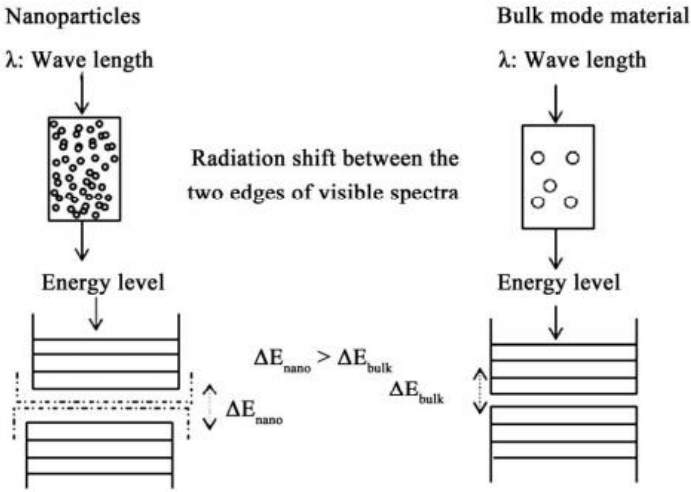
2.6 Quantum Dot

Quantum dot (QD) atau disebut juga carbon dot ialah sebuah bahan semikonduktor memiliki ukuran pada skala nanometer dan bereksitasi pada skala tiga dimensi yang memiliki bahan utama yaitu senyawa organik. QD memiliki sifat optis dan elektris yang dapat diatur dengan cara mengatur ukuran dari nanopartikel QD tersebut, sehingga QD memiliki aplikasi yang luas dalam bidang nanoteknologi [13].



Gambar 2.9 Ukuran *quantum dot* diurutkan dari yang terkecil (warna ungu) hingga terbesar (merah) [13].

Berbeda dengan *bulk semiconductor,* CQD memiliki ukuran lebih kecil dari 10nm, meski keduanya merupakan nanomaterial dengan eksitasi tiga dimensi. Konsekuensi dari hal tersebut dapat dilihat dari perbedaan ukuran yang dimiliki oleh bahan *bulk semiconductor* maupun *quantum dot* terhadap warna dari larutan tersebut, seperti pada gambar 2.9.



Gambar 2 10 Perbedaan *band gap energy* antara bahan *bulk semiconductor* dengan nanopartikel QD [14].

Band-gap energy yang merupakan lebar antara tingkat energi satu dengan yang lainnya seperti pada gambar 2.10 yang pada suatu QD didefinisikan melalui persamaan Brus [14]:

$$E_{g(QD)} = E_{bulk}\frac{h^2}{8R^2}\left(\frac{1}{m^*_e} + \frac{1}{m^*_h}\right) - \frac{1.76e^2}{4\pi\varepsilon_0\varepsilon_r R^2} \tag{2.28}$$

Di mana $E_{g(QD)}$ merupakan *band gap energy* dari bahan QD, E_{bulk} merupakan *band gap energy* dari bahan *bulk semiconductor*, h merupakan konstanta Planck, R merupakan jari-jari atom, e adalah muatan electron, ε_0 dan ε_r masing-masing merupakan permitivitas vakum dan bahan, dan m^*_e dan m^*_h masing-masing merupakan massa efektif dari eksitasi electron dan *hole*. Untuk mencari energi emisi bahan QD, persamaan (2.28) dapat dimodifikasi sebagai berikut:

$$\Delta E = E_g(R) + \frac{h^2}{8R^2}\left(\frac{1}{m^*_e} + \frac{1}{m^*_h}\right) \tag{2.29}$$

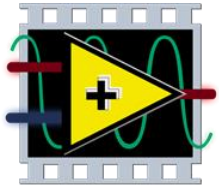
Quantum dot terdiri atas dua jenis, tergantung dari jenis bahan yang digunakan. Bentuk QD yang paling umum adalah QD berbasis bahan semikonduktor, seperti cadmium selenida (*Cadmium Selenide/*CdSe) yang dibuat dengan melakukan sintesis antara cadmium oxide (CdO) dengan unsur selenium. Adapun bahan QD juga dapat terbuat dari bahan organik, atau disebut dengan *carbon quantum dot*/*carbon dot*.

Aplikasi dari QD sendiri dapat ditemukan pada bidang energi, di mana *quantum dot* CdSe digunakan dalam pembuatan sel fotovoltaik. Penggunaan QD sendiri, karena sifat optisnya yang mudah diubah, dapat digunakan untuk membuat LED warna putih yang lebih efisien.

2.7 NI LabVIEW 2018

National Instrument Laboratory Virtual Instrument Engineering Workbench (LabVIEW 2018), dengan logo pada gambar 2.11, merupakan software yang berfungsi sebagai pemroses visual dalam bidang akuisisi data, *instrumentation control* serta automasi industri yang pertama kali dikembangkan oleh perusahaan National Instruments pada tahun 1986. Perangkat lunak ini dapat dijalankan pada sistem operasi Linux, Unix, Mac OS X dan Windows[15].

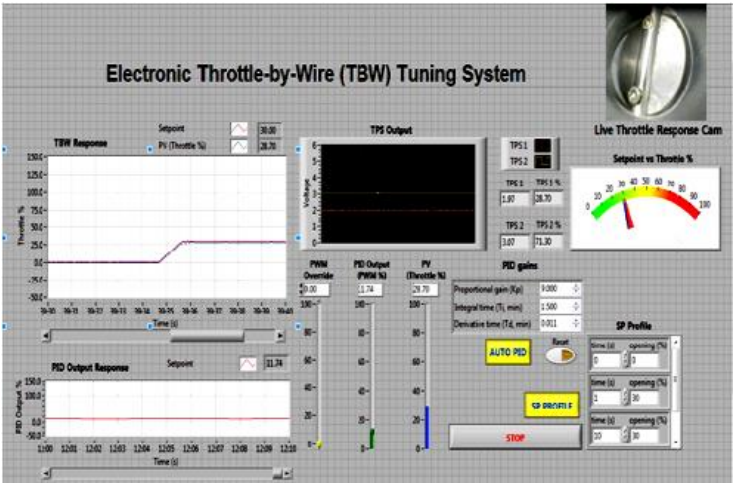
.



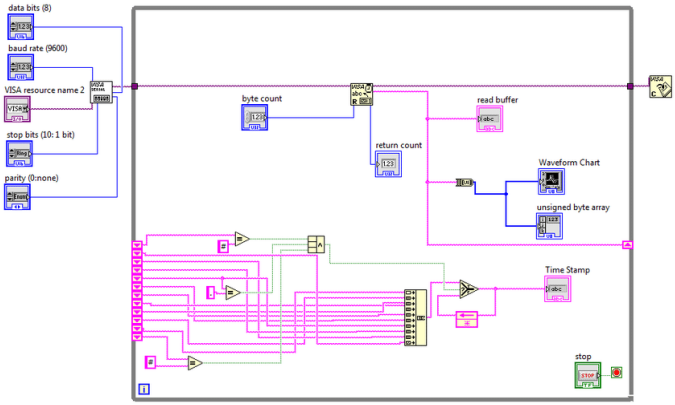
Gambar 2.11 Logo LabVIEW [15].

LabVIEW memiliki *program* subroutine yang disebut sebagai Virtual Instrument (VI). Setiap VI memiliki tiga komponen: diagram blok, panel depan, dan panel konektor yang terakhir digunakan untuk mewakili VI dalam diagram blok lainnya, memanggil VI. Panel depan dibangun menggunakan kontrol dan indikator seperti yang ditunjukkan pada gambar 2.12 dan 2.13. Panel kontrol adalah memungkinkan

pengguna untuk memberikan informasi kepada VI. Panel indikator adalah output yang berfungsi untuk menunjukkan, atau menampilkan, hasil berdasarkan input yang diberikan kepada VI.



Gambar 2.12 Control Panel dari LabVIEW [15].



Gambar 2.13 Contoh Block Diagram Panel dari LabVIEW [15].

Panel belakang, yang merupakan diagram blok, berisi kode sumber grafis. Semua objek yang ditempatkan di panel depan akan muncul di panel belakang sebagai terminal. Panel belakang juga berisi struktur dan fungsi yang melakukan operasi pada kontrol dan memasok data ke indikator. Struktur dan fungsi ditemukan pada palet Functions dan dapat ditempatkan pada panel belakang. LabVIEW menggunakan bahasa G dalam pemrogramannya. Bahasa G tidak menggunakan teks dan hanya menyediakan pemrograman secara visual.

Salah satu fitur utama dalam LabVIEW adalah adanya Virtual Instrument Software Architecture (VISA). VISA merupakan *application programming interface* (API) yang digunakan untuk mengontrol input dari luar, seperti *ethernet*, USB, dsb.VISA memungkinkan pengguna untuk mengontrol penuh input eksternal yang dipasangkan pada komputer.

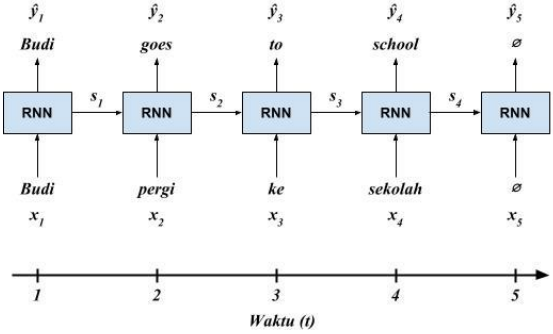
2.8 Recurrent Neural Network & LSTM

Recurrent Neural Network merupakan arsitektur *neural network* yang ditujukan untuk mengklasifikasi data secara berulang-ulang untuk memroses input yang biasanya adalah data sekuensial seperti data *time-series*, yakni data yang berubah terhadap waktu [16].

Keunggulan RNN dibandingkan metode neural network lain seperti *multi-layer perceptron* ataupun *convolutional neural network* adalah kendala yang dimiliki sistem yang disebutkan[17]:

1) Model yang dipakai untuk CNN maupun MLP tidak memiliki memori, sehingga pemrosesan sample tidak memperhitungkan sample pada sekuen sebelumnya.
2) Karena model tersebut tidak memiliki memori, data yang diinputkan pada model diasumsikan sebagai independen, berbeda dengan data *time-series* yang seringkali memiliki hubungan antar satu sama lain.
3) Model selain RNN memroses masukan dengan panjang input yang tetap, sedangkan data sekuensial memiliki interval yang kadang berbeda-beda
4) Model MLP maupun CNN tidak bisa meng-klasifikasikan fitur secara fleksibel.

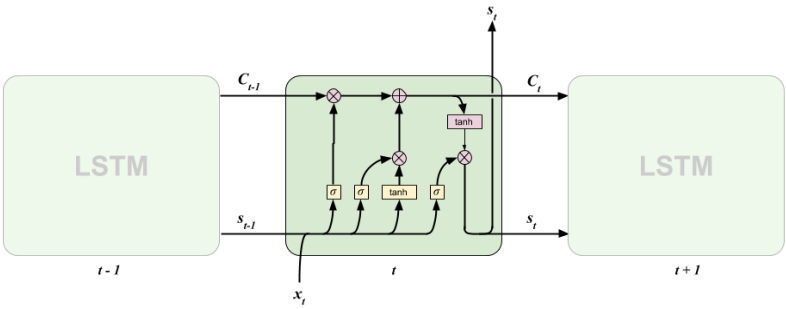
RNN memroses input secara runtun. Dalam setiap prosesnya, output yang dihasilkan didasarkan pada pemrosesan sampel-sampel sebelumnya. Adapun pemrosesan ini juga dilakukan pada sampel setelahnya pada kasus *bidirectional RNN*. Ilustrasi algoritma RNN dapat dilihat pada gambar 2.14 [18].



Gambar 2.14 Diagram blok sistem RNN [18]

Input *x* merupakan input data *time-series*/sekuensial yang hendak diproses ke dalam RNN. Dalam sebuah sel RNN, data sekuensial dipecah menjadi beberapa bagian. Adapun hasil pemrosesan data dalam satu sel tersebut akan dikeluarkan sebagai output *y* dan umpan balik *s* untuk sel RNN selanjutnya.

Terdapat beberapa jenis arsitektur yang populer dalam RNN, yakni LSTM/Long Short Term Memory ataupun Gated Recurrent Unit. LSTM merupakan salah satu dari arsitektur RNN yang terdiri atas sel, gerbang input, gerbang *forget* dan gerbang output[19].



Gambar 2.15 Visualisasi salah satu blok LSTM [19]

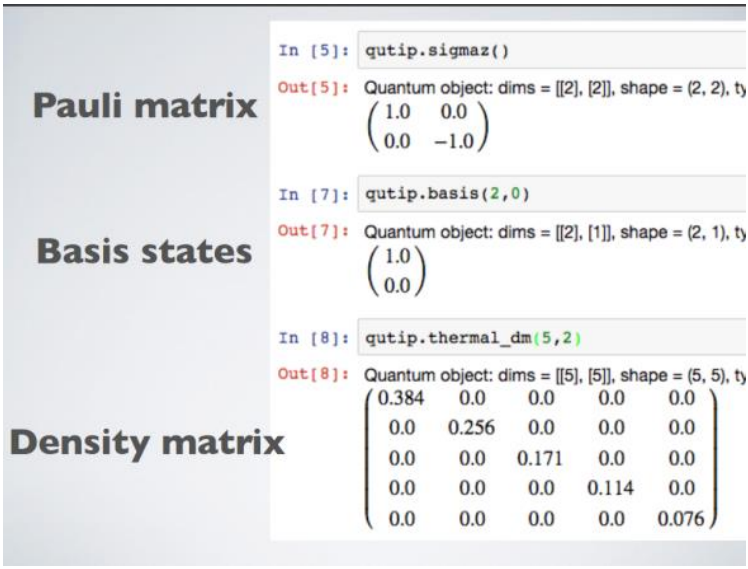
Sesuai pada gambar 2.15, LSTM memiliki gerbang *C(t-1)* sebagai input data dari waktu sebelum *C(t)* yang menjadi input untuk LSTM selanjutnya [20]. LSTM memiliki satu gerbang bernama gerbang *forget* yang mengeluarkan angka 0 atau 1 untuk menentukan data *C(t-1)* mana yang perlu disimpan berdasarkan input *s(t-1)* dan *x(t)*. Untuk setiap nilai *C(t-1)* yang dilewatkan, nilai tersebut akan diproses bersamaan dengan input *s* dan *t* yang dimasukkan ke dalam fungsi aktivasi, yang secara umum merupakan fungsi *sigmoid* ataupun *tanh*. Setelah pemrosesan tersebut, data akan dikeluarkan ke dua bagian, yakni output *C(t)* yang dikeluarkan tanpa pemrosesan lagi seperti output *s(t)* yang perlu diproses lagi dalam fungsi aktivasi tanh. *C(t)* digunakan untuk input sel LSTM selanjutnya beserta dengan *s(t)*, namun nilai *s(t)* juga digunakan sebagai output yang ditampilkan dari sistem.

2.9 QuTiP

Quantum Toolbox in Python, atau disingkat QuTiP merupakan software open-source untuk mensimulasikan dinamika sistem kuantum terbuka dengan logo pada gambar 2.16 [20]. Library QuTiP bergantung pada library Numpy, Scipy, dan Cython. Selain itu, output grafis disediakan oleh Matplotlib. QuTiP bertujuan untuk menyediakan simulasi numerik yang ramah pengguna dan efisien dari berbagai Hamiltonian, termasuk yang dengan ketergantungan waktu sewenang-wenang, umumnya ditemukan dalam berbagai aplikasi fisika seperti optik kuantum, trapped ion, superkonduktor, dan resonator nanomekanik kuantum . QuTiP tersedia secara bebas untuk digunakan dan / atau dimodifikasi pada semua platform utama seperti Linux, Mac OSX, dan Windows.

Gambar 2.16 Logo QuTiP [20].

QuTiP mengemas semua operator yang diperlukan dalam simulasi mekanika kuantum menjadi satu library. Operator dalam persamaan mekanika kuantum telah disediakan dalam bentuk matriks seperti yang ditampilkan pada gambar 2.17.



Gambar 2.17 Beberapa matriks mekanika kuantum dasar dalam QuTip [20]

2.10 TensorFlow

Tensorflow, dengan logo pada gambar 2.18, adalah sebuah library open source gratis yang digunakan untuk dataflow, yang merupakan struktur untuk pengaliran data di dalam sistem pemrosesan

data. Aplikasi dari TensorFlow banyak ditemukan pada *machine learning* seperti *neural network* yang dikembangkan oleh tim Google Brain [21].

Gambar 2.18 Logo TensorFlow [21].

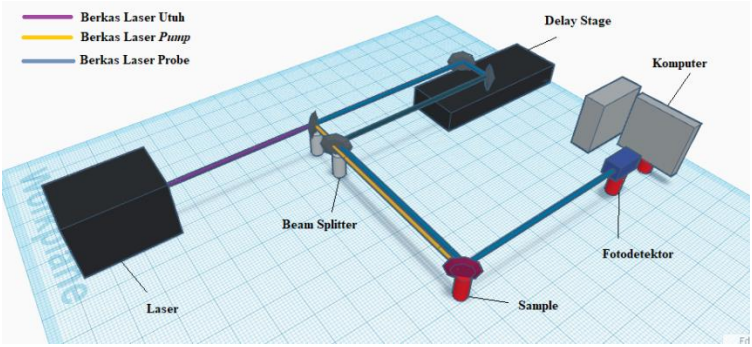
Halaman ini sengaja dikosongkan.

BAB 3
PERANCANGAN SISTEM

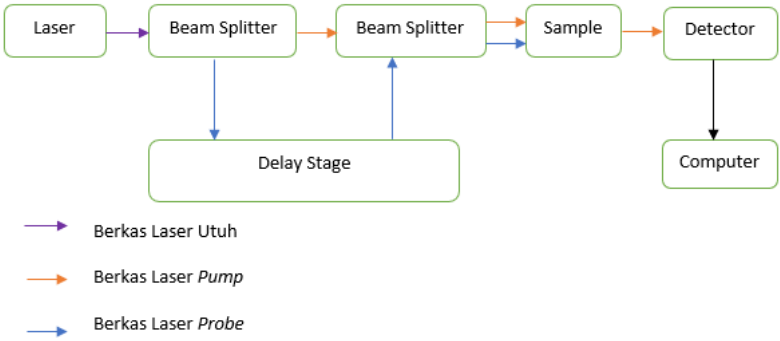
Pada bab ini dijelaskan perancangan sistem secara keseluruhan. Perangkat yang dirancang bertujuan untuk melakukan simulasi dari interaksi antara laser femtosekon dengan senyawa yang dituju beserta akuisisi data dari sistem tersebut, serta klasifikasi dari data sample yang telah didapatkan. Simulator inipun ditunjang dengan pengambilan data eksperimental yang dilakukan di Laboratorium Femtosekon, Pusat Penelitian Fisika – LIPI, sebagai penghubung antara teori dan eksperimen sistem *pump-probe spectroscopy*. Simulator sistem *pump-probe* diprogram dengan Python dengan library QuTiP sebagai penunjang yang memberikan definisi persamaan fisika kuantum yang dibutuhkan dalam pembuatan simulator. Sedangkan dalam sisi eksperimental, data dari sample yang diakuisisi melalui software NI LabVIEW 2018, dan data tersebut dilatih melalui *Recurrent Neural Network*

3.1 Perancangan Elektronik Sistem

Sistem ini terdiri atas tiga jenis sub-sistem, yakni akuisisi data, simulator, serta klasifikasi data. Akuisisi data sendiri masuk ke dalam perancangan perangkat keras, simulator dan klasifikasi data masuk ke dalam perancangan perangkat lunak. Visualisasi dari perancangan perangkat keras serta diagram blok sistem dapat dilihat pada gambar 3.1 dan 3.2.



Gambar 3.1 Visualisasi Sistem *Pump-probe Spectroscopy*.



Gambar 3.2 Blok Diagram Sistem *Pump-probe Spectroscopy*

3.2 Perancangan Perangkat Keras

Pada tahap ini dilakukan perancangan perangkat keras berupa penyusunan *setup* laser pulsa skala femtosekon, yakni terdiri atas:

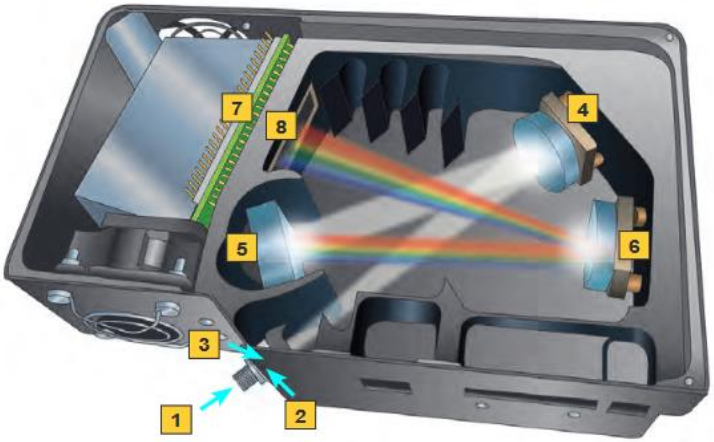
1) Laser
   Sumber laser dengan pulsa pada skala femtosekon $(10^{-15}\ \mathrm{s})$. Memiliki panjang gelombang 1064 nm.
2) Beam Splitter
   Pemecah berkas laser menjadi dua bagian, yakni *pump* dan *probe*.
3) Delay Stage Newport DL125
   Delay Stage Newport DL125 mengatur besar *delay* dari berkas laser *probe* dengan menggerakan cermin pemantul menjauhi sumber laser.
4) Sample
   Bahan uji yang akan dianalisis. Sinyal *pump* dikenakan pada *sample* untuk dieksitasi dan sinyal probe untuk menganalisis bahan tersebut.
5) Fotodetektor
   Sebagai alat untuk mengakuisisi data dari berkas laser *probe*. Menggunakan spektrofotometer MayaPRO2000.
6) Komputer
   Untuk memroses data yang telah ditangkap oleh fotodetektor.

Adapun tahap ini juga meliputi *troubleshooting* bila terjadi kesalahan desain.

Akuisisi data ultra-cepat dilakukan dengan merekam jejak spectra intensitas yang diakuisisi oleh spektrofotometer MayaPro2000 [22]. Komponen yang terdaftar pada gambar 3.3 adalah sebagai berikut:

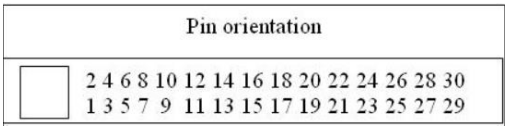
1) SMA905 Connector
   Sebagai konektor *fibre optic* ke dalam spektrofotometer
2) Kisi Masukan
   Untuk mengontrol cahaya yang masuk dari konektor.
3) Longpass Filter
   Untuk menyaring panjang gelombang yang tidak diinginkan, dengan pilihan filter pada panjang gelombang di bawah 305 nm sampai 590 nm.
4) Cermin Kolimasi
   Untuk meng-kolimasi cahaya yang sudah keluar dari filter, dan kemudian akan disalurkan ke kisi.
5) Kisi
   Kisi yang dapat dimodifikasi untuk menentukan panjang gelombang dari cahaya yang masuk.
6) Cermin FokusMemfokuskan cahaya yang sudah keluar dari kisi agar tidak terhamburkan.
7) Detektor Hamamatsu
   Untuk mendeteksi cahaya masuk dengan resolusi pixel 2048 x 64. Memiliki *integration time* sebesar 7.2ms - 5 sekon serta efisiensi kuantum lebih dari 90%.
8) OFLV Filter
   Pada optika non-linier, digunakan untuk memblokir cahaya orde dua maupun tiga.

Data spectra yang didapatkan oleh MayaPro2000, sebagaimana yang ditunjukkan pada mekanisme gambar 3.3, masih berupa intensitas per panjang gelombang, sehingga diperlukan sedikit modifikasi untuk mengambil data intensitas terhadap waktu, dengan input berupa panjang gelombang yang diinginkan.



Gambar 3.3 Visualisasi spektrofotometer MayaPro2000 [22].

Detail pinout dari spektrofotometer MayaPro2000 telah ditampilkan pada gambar 3.4. Konektor yang digunakan untuk menyambung semua pin ini ialah Pak50TM . Pin 1 dan 2 diperuntukkan masing-masing untuk *transmitter/receiver* RS232. Sedangkan pin 3, 7, 9, 11, 16, 18, 22, 26, 28, dan 30 digunakan sebagai pin GPIO. Pin 4, 13, 19, 21, 23, dan 24 tidak terkoneksi pada perangkat spektrofotometer. Untuk input VCC terdapat pada pin 12 dan 14, sedangkan untuk ground terdapat pada pin 5, 27 dan 29. Spektrofotometer MayaPRO2000 juga mendukung penggunaan I2C dengan pin clock dan data yang terdapat pada nomor 6 dan 8. Adapun input lainnya berupa input TTL trigger pada pin 10 dan input SPI data pada pin 15. Sedangkan output untuk stroboscope terdapat pada pin 17 dan 20.



.

Gambar 3.4 Pinout spektrofotometer MayaPro2000 [22]

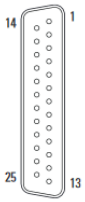
Gambar 3.5 Visualisasi *delay stage* Newport DL325 [23].

Pada gambar 3.5 juga dicantumkan *pinout* dari delay stage Newport DL325. Pada konektor motor, terdapat tiga jenis motor, yakni motor U, V, dan W yang direpresentasikan pada pin 1, 2, 5, 6, 7, dan 8. Motor tersebut digunakan untuk mengatur *roll*, *pitch*, dan *yaw* dari pergerakan motor pada Newport DL325. Terdapat 12 pin yang tidak memiliki koneksi, yakni pin 3, 4, 10, 11, 12, 13, 25, 16, 19, 20, 23, 24, dan 25.Pin VCC terdapat pada pin 21 dan ground terdapat pada pin nomor 14, 16, dan 22. Ada juga pin yang disediakan untuk thermistor, *service management bus* (SMB) clock dan data, yang masing-m,asing terdapat pada pin 9, 17, dan 18.

Sedangkan pada konektor encoder, terdapat pin berupa sine, cosine, dan index pulse I yang digunakan sebagai *feedback signal position* pada posisi delay stage, yang digunakan untuk mengetes apakah delay stage merespon perintah yang diberikan user atau tidak dengan pergerakan sesuai dengan sinyal yang diberikan. Pin untuk sinyal-sinyal tersebut terdapat pada pin 4, 6, dan 8. Sedangkan untuk *inverse*/kebalikan dari sinyal tersebut terdapat pada pin 13, 15, dan 17. Pin 5 dan 14 menandakan posisi *end-of-run* atau posisi terakhir dari delay stage. Pin yang tersisa merupakan pin VCC dan pin ground, masing-masing tedapat pada pin 1, 19, serta 7, 25, dan 26.

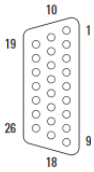
Adapun delay yang diinginkan pada pengukuran disesuaikan dengan tahapan pergerakan pada delay stage DL125, yang dimana kecepatan per tahapan gerakan dapat diatur juga, sesuai pada gambar 3.4 [24]. Dengan jarak tempuh hingga 125 mm ± 1.5 µm dan jarak minimum pergerakan sebesar 75 nm , DL125 mampu memberikan resolusi *delay* yang sangat tinggi pada cahaya laser *probe*.

Motor Connector (SUB-D25M)



1	U Motor	14	Ground
2	U Motor	15	N.C.
3	N.C.	16	Ground
4	N.C.	17	Reserved (SMDAT)
5	V Motor	18	Reserved (SMCLK)
6	V Motor	19	N.C.
7	W Motor	20	N.C.
8	W Motor	21	+5 V
9	Reserved (Thermistor)	22	Ground
10	N.C.	23	N.C.
11	N.C.	24	N.C.
12	N.C.	25	N.C.
13	N.C.		

Encoder Connector (HD26M)



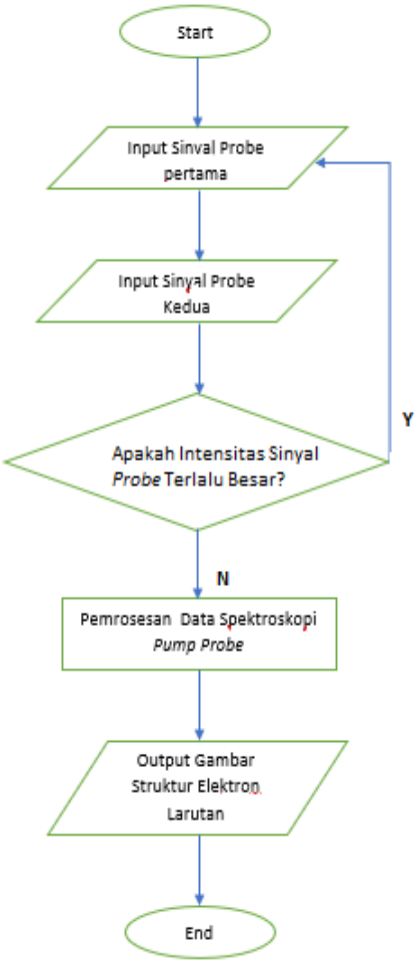
1	+5 V	14	+ End-of-Run
2	N.C.	15	/Sine
3	N.C.	16	N.C.
4	Cosine	17	Index Pulse /I
5	- End-of-Run	18	N.C.
6	Sine	19	+ 5 V
7	Ground	20	N.C.
8	Index Pulse I	21	N.C.
9	N.C.	22	N.C.
10	N.C.	23	N.C.
11	N.C.	24	N.C.
12	N.C.	25	Ground
13	/Cosine	26	Ground

Gambar 3.6 Diagram *pinout* motor dan encoder pada Newport DL325 [23]

3.3 Perancangan Perangkat Lunak Sistem

3.3.1 Perancangan Sistem Akuisisi Data

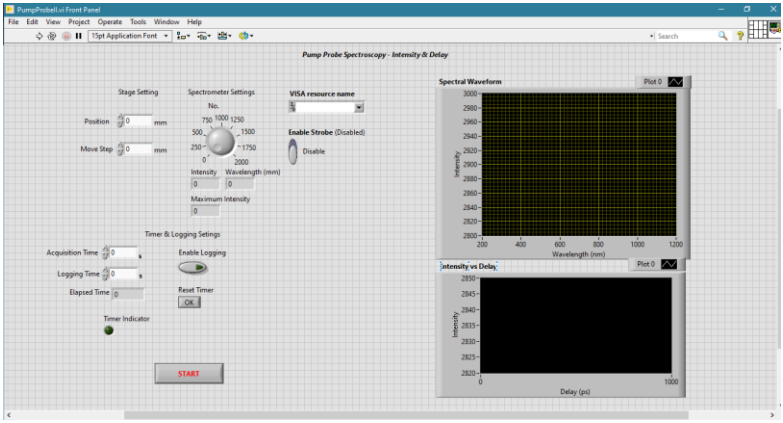
Pada tahap ini dilakukan perancangan perangkat lunak yang meliputi *setup* program yang digunakan dalam akuisisi data dari sistem *Pump-probe* dengan menggunakan NI LabVIEW 2018. Diagram blok dari sistem dapat dilihat pada gambar 18. Setelah itu, data-data yang telah didapatkan dari sistem tersebut akan diproses melalui metode *neural network* agar dapat mempermudah prediksi data-data yang akan diakuisisi selanjutnya. Diagram alur/*flowchart* dari proses akuisisi data akan dijelaskan dalam gambar 3.6.



Gambar 3.5 Diagram Alir Keseluruhan Sistem Akuisisi Data.

Pada gambar 3.6, terdapat dua jenis proses yang berjalan di dalam perangkat lunak akuisisi data LabVIEW, yakni akuisisi data dari spektrofotometer dan penggerak *delay stage*. Tampilan grafis juga

dibuat untuk memudahkan pengguna dalam menggunakan sistem akuisisi data, di mana terdapat output berupa data spektra sinyal *input* untuk memudahkan pengguna program melihat spektrum panjang gelombang dengan intensitas tertinggi. dan data perubahan intensitas terhadap *delay* waktu pada sinyal *probe*, sebagaimana yang terlihat pada gambar 3.7.



Gambar 3.6 *Graphical User Interface* akuisisi data sistem *pump-probe* pada program LabVIEW.

Akuisisi data spektrofotometer memiliki tiga jenis input, yakni input alamat instrumen pada sistem operasi yang dilabeli VISA yang berisi data alamat dari spektrofotometer untuk memasukkan input intensitas cahaya, input panjang gelombang untuk menampilkan sinyal pada panjang gelombang yang berbeda, input *Boolean* bernama Enable Logging, Reset Timer dan Enable Strobe, yang masing-masing berfungsi untuk menyalakan *data logging* pada output *Intensity vs delay* , mengulangi waktu perekaman dari sample, dan mengaktifkan *stroboscope* pada spektrofotometer, serta yang terakhir ialah input numerik *acquisition time* dan *logging time* yang masing-masing menentukan waktu total akuisisi data dan rentan waktu dari awal sampai akhir akuisisi data pada sistem.

Sedangkan untuk sistem *delay stage* terdapat input numerik berupa *position* dan *move step* dan input *Boolean* berupa *Start*. Input posisi digunakan untuk menentukan posisi awal dari *delay stage* dan

move step digunakan untuk menentukan kelajuan dari *delay stage* dari awal sampai akhir. *Start* digunakan untuk memulai akuisisi data sistem dan penggerakan *delay stage*.

Adapun dalam setup eksperimen ini digunakan dua jenis sample yakni larutan *carbon dot* lengkuas dan jahe. Data diakusisi dengan *acquisition time* sebesar 0.5 detik dan *move step* sebesar 2 mm, setara dengan pergerakan stage dari delay 0 pikosekon hingga 13 pikosekon.

3.3.2 Perancangan Sistem Simulator

Simulator dirancang dengan menggunakan library QuTiP pada Python. Pemrograman simulator dimulai dengang meng-import library yang diperlukan untuk simulasi, seperti contohnya qutip, numpy, dsb. Sebagaimana ditunjukkan pada gambar 3.7

```
In [1]: from qutip import *
        import numpy as np
        import matplotlib.pyplot as plt
```

Gambar 3.7 Mengimpor package yang diperlukan

Setelah itu perlu juga mendefinisikan matriks-matriks *ket* utama, yakni matriks *ground state* bernama |g>, *excited state* bernama |e>, dan matriks *steady state* bernama |u> yang didefinisikan sebagai demikian:

$$|g> = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \tag{3.1}$$

$$|e> = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \tag{3.2}$$

$$|u> = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{3.3}$$

Langkah kedua ialah membangun matriks produk |g><e| dan |u><e| yang digunakan untuk menandakan transisi dari *state* dasar ke eksitasi

dan ke *steady state*. Matriks ini secara matematis direpresentasikan sebagai:

$$\sigma_{ge} = I(2,3\ ) \otimes |g><e| \tag{3.4}$$

$$\sigma_{ue} = I(2,3) \otimes |g><e| \tag{3.5}$$

Di mana $I(2,3\ )$ merupakan matriks identitas yang digunakan untuk memekarkan dimensi produk $|g><e|$ dan $|u><e|$ dari 3 x 3 menjadi matriks 6 x 6.Hal ini akan meningkatkan akurasi yang didapatkan pada hasil simulasi.

Langkah keempat ialah mendefinisikan matriks *collapse operator* yang digunakan untuk memodelkan disipasi energi dalam simulasi *pump-probe.* Konstanta yang ditentukan dalam *collapse operator* ialah *decay rate* dari rongga optik (dan dari atom (γ_n) yang diuji. Di sini didefinikan matriks *a*:

$$a = destroy(2,2) \otimes I(3,3\ ) \tag{3.6}$$

Di mana *destroy(2,2)* merupakan operator penurun *eigenvalue* dari suatu sistem kuantum yang dimekarkan melalui perkalian tensor. Operator *a* kemudian akan dimasukkan ke dalam persamaan (25) bersamaan dengan σ_{ge} dan σ_{ue}.

Setelah itu, untuk dapat menampilkan data melalui grafik, diperlukan operator proyektor |g> <g| dan |u> <u| yang didefinisikan sebagai:

$$\sigma_{GG} = \binom{0}{1} \otimes |g> * (\binom{0}{1} \otimes |g>)^{\dagger} \tag{3.7}$$

Begitu juga untuk *steady state*:

$$\sigma_{UU} = \binom{0}{1} \otimes |U> * (\binom{0}{1} \otimes |U>)^{\dagger} \tag{3.8}$$

Kemudian dari sini dapat disusun Hamiltonian dari sistem. Terdapat tiga jenis Hamiltonian, yakni Hamiltonian yang tidak terpengaruh waktu (H1) yang merupakan persamaan (2.28), dan dua

yang terpengaruh waktu (H2) dan (H3). Khusus untuk Hamiltonian terpengaruh waktu seperti H2 dan H3 akan dikalikan dengan koeffisien bergantung waktu lainnya, yakni medan listrik yang dipancarkan dari sinyal laser *pump* dan *probe*. Persamaan medan listrik dari laser dapat didefinisikan sebagai bentuk *Gaussian beam.*

$$E_{pump(\tau_3,\ \tau_2)} = \frac{E_0^2}{2\pi\sigma_a^2} e^{\frac{(\tau_3-\tau_a)^2}{\sigma_a^2}} e^{\frac{(\tau_3-\tau_3)^2}{\sigma_a^2}} \tag{3.9}$$

$$E_{probe(\tau_2,\ \tau_1)} = \frac{E_0^2}{2\pi\sigma_b^2} e^{\frac{(\tau_2-\tau_b)^2}{\sigma_b^2}} e^{\frac{(\tau_2-\tau_1)^2}{\sigma_b^2}} \tag{3.10}$$

Setelah semua paramter telah didefinisikan, Hamiltonian sistem dapat digabung menjadi satu sebelum dimasukkan ke dalam fungsi mesolve().

3.3.3 Perancangan Sistem Klasifikasi Data

Pada tahap perancangan klasifikasi sistem, kita perlu mendefinisikan library yang akan dipakai, seperti numpy dan tensorflow, dan pandas. Kemudian dengan menggunakan pandas kita mengambil dataset yang akan dijadikan acuan latihan dari RNN:

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from matplotlib import rc
from pandas.plotting import register_matplotlib_converters
from sklearn.preprocessing import RobustScaler
%matplotlib inline
%config InlineBackend.figure_format='retina'

register_matplotlib_converters()
sns.set(style='whitegrid', palette='muted', font_scale=1.5)

rcParams['figure.figsize'] = 22, 10

RANDOM_SEED = 42

np.random.seed(RANDOM_SEED)
tf.random.set_seed(RANDOM_SEED)
```

Gambar 3.8 Mendefinisikan *library* yang diperlukan

Setelah dataset diupload, terlihat pada gambar 3.9 bahwa dataset tersebut dibagi menjadi dua bagian, yakni data training dan data test. Kemudian, kedua jenis data tersebut diskala ulang dan dikonversi menjadi *array* numpy seperti pada gambar 3.10:

```
def create_dataset(X, y, time_steps=1, step=1):
    Xs, ys = [], []
    for i in range(0, len(X) - time_steps, step):
        v = X.iloc[i:(i + time_steps)].values
        labels = y.iloc[i: i + time_steps]
        Xs.append(v)
        ys.append(stats.mode(labels)[0][0])
    return np.array(Xs), np.array(ys).reshape(-1, 1)
```

Gambar 3.9 Melakukan skala ulang pada dataset dengan mendefinisikan fungsi *create_dataset*

Proses ini dinamakan sebagai *preprocessing*. Setelah melakukan skala ulang, dataset perlu dibentuk ulang untuk dapat dimasukkan ke dalam layer LSTM seperti pada gambar 3.11

```
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(handle_unknown='ignore', sparse=False)

enc = enc.fit(y_train)

y_train = enc.transform(y_train)
y_test = enc.transform(y_test)
```

Gambar 3.10 Pembentukan ulang dataset sesuai dengan data training dan test menggunakan encoder.

Dengan demikian, tahap *preprocessing* telah selesai. Tahap yang selanjutnya dilakukan ialah membangun model yang diinginkan, yakni model LSTM dengan 128 layer. Setelah LSTM tersebut dipasangkan *droupout layer* yang digunakan untuk mencegah overfitting, Dense layer yang berisi masing-masing fungsi aktivasi ReLU dan Softmax. Langkah terakhir yang dapat dilakukan sebelum melakukan training adalah mengkonfigurasi loss yang diperlukan dengan fungsi model.compile().

```
model = keras.Sequential()
model.add(
    keras.layers.Bidirectional(
      keras.layers.LSTM(
          units=128,
          input_shape=[X_train.shape[1], X_train.shape[2]]
      )
    )
)
model.add(keras.layers.Dropout(rate=0.5))
model.add(keras.layers.Dense(units=128, activation='relu'))
model.add(keras.layers.Dense(y_train.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
```

Gambar 3.11 Visualisasi pembangunan model *neural network* yang diperlukan

Sesuai dengan gambar 3.12, training dilakukan dengan menggunakan layer LSTM sebanyak 128 buah. Kemudian training dilakukan masing-masing sebanyak 100 dan 12000 epos untuk kemudian data test dapat digunakan untuk mengetes akurasi model yang telah dilatih. Visualisasi prediksi data dilakukan menggunakan *confusion matrix* pada gambar 3.13.

```
from sklearn.metrics import confusion_matrix

def plot_cm(y_true, y_pred, class_names):
  cm = confusion_matrix(y_true, y_pred)
  fig, ax = plt.subplots(figsize=(12, 10))
  ax = sns.heatmap(
      cm,
      annot=True,
      fmt="d",
      cmap=sns.diverging_palette(220, 20, n=7),
      ax=ax
  )

  plt.ylabel('Actual')
  plt.xlabel('Predicted')
  ax.set_xticklabels(class_names)
  ax.set_yticklabels(class_names)
  b, t = plt.ylim()
  b +=
  t -=
  plt.ylim(b, t)
  plt.show()
```

Gambar 3.12 Script Python untuk visualisasi data hasil prediksi

Perhitungan untuk akurasi dari *confusion matrix* diberikan melalui persamaan berikut:

$$Akurasi = \frac{True\ Positive + True\ Negative}{Total\ percobaan} \quad (3.11)$$

Di mana *true positive* merupakan nilai di mana hasil prediksi benar sesuai dengan hasil yang benar, *true negative* adalah nilai di mana hasil prediksi salah sesuai dengan hasil yang salah.

Halaman ini sengaja dikosongkan.

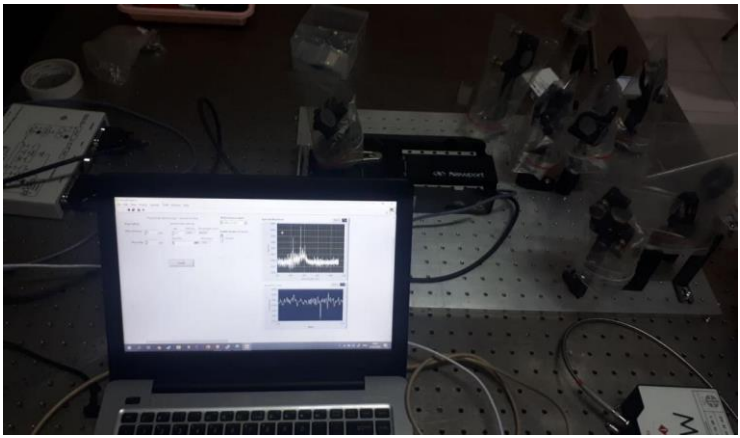
BAB 4
PENGUJIAN DAN ANALISIS

Pada bab ini akan dijelaskan mengenai pengujian analisis perangkat keras dan perangkat lunak dari sistem yang telah dibangun pada bab sebelumnya. Pada bab ini juga disusun untuk memperoleh analisis data dari setiap pengujian yang telah dilakukan. Pengujian dilakukan per sistem yang telah dikerjakan, mulai dari perangkat keras, perangkat lunak berupa simulator dan perangkat lunak berupa sistem klasifikasi data.

4.1 Pengujian Setup Eksperimen

4.1.1 Pengaturan Setup Perangkat Keras

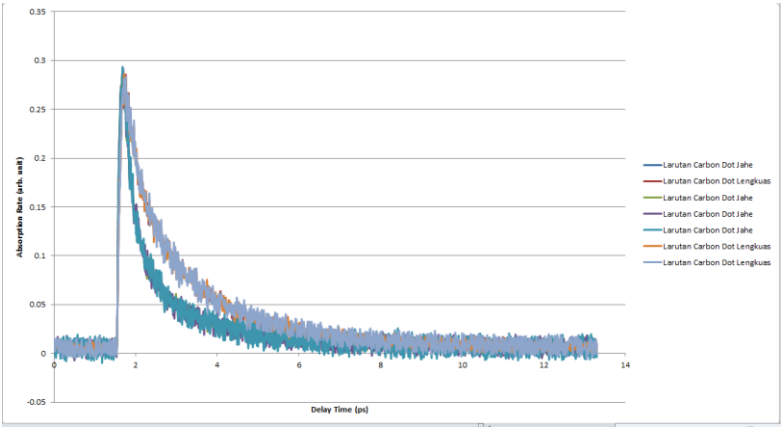
Pada pengujian perangkat keras, digunakan perangkat yang tersedia di Laboratorium Femtosekon PPF LIPI yang telah menyediakan perangkat seperti yang telah dijelaskan pada bab 3. Dokumentasi dari percobaan yang telah dilakukan dapat dilihat pada gambar 4.1



Gambar 4.1 Visualisasi Setup Eksperimen dan Pengambilan Data

4.1.2 Pengujian Sistem Akuisisi Data

Gambar 4.2 menunjukkan hasil dari data yang telah diakuisisi. Data diambil dengan perekaman spektrofotometer sekitar 0.5 sekon dan penambahan waktu *delay* dari 0 sampai 13 pikosekon.



Gambar 4.2 Grafik Akuisisi data sistem *pump-probe spectroscopy*

Dari gambar 4.2 terlihat bahwa data yang diambil dari sistem akuisisi data memiliki karakteristik yang cukup berbeda meski memiliki jenis yang kurang lebih sama. Grafik sumbu x merupakan waktu delay dari sample yang terkena laser *probe* dalam satuan pikosekon, sedangkan grafik sumbu y menandakan nilai perubahan absorbansi dari sample yang dibuat. Terlihat bahwa larutan *carbon dot* pada lengkuas memiliki nilai absorbansi yang lebih tinggi dibandingkan dengan larutan *carbon dot* jahe. Data yang ditampilkan pada gambar 4.2 merupakan 7 data pertama dari 40 data yang telah diambil dari setup eksperimen, di mana masing-masing sample terdapat 20 kali pengambilan data.

4.2 Pengujian Simulator

4.2.1 Implementasi persamaan menggunakan QuTiP

Pada tahap ini, implementasi dari persamaan 3.1 sampai 3.8 telah dilakukan. Persamaan mekanika kuantum yang telah didefinisikan secara teoretis diimplementasikan melalui fungsi yang disediakan pada

library QuTiP. Implementasi dari hal tersebut dapat dilihat pada gambar 4.3 pada aplikasi Jupyter Notebook.

Gambar 4.3 Pemanggilan *library* untuk penulisan matriks *ground state*, *excited state*, dan *steady state*.

Adapun matriks *ground*, *excited*, dan *steady state* didefinisikan sebagaimana ditunjukkan pada gambar 4.3. Perkalian tensor pada matriks dengan mudah dapat dilakukan dengan memanggil fungsi tensor() pada gambar 4.4.

Gambar 4.4 Perkalian Tensor pada matriks *ground*, *excited*, dan *steady state*

Langkah terakhir yang dilakukan adalah mendefinisikan Hamiltonian sistem, matriks proyektor, dan operator *collapse* pada simulator sebagaimana ditunjukkan pada gambar 4.5.

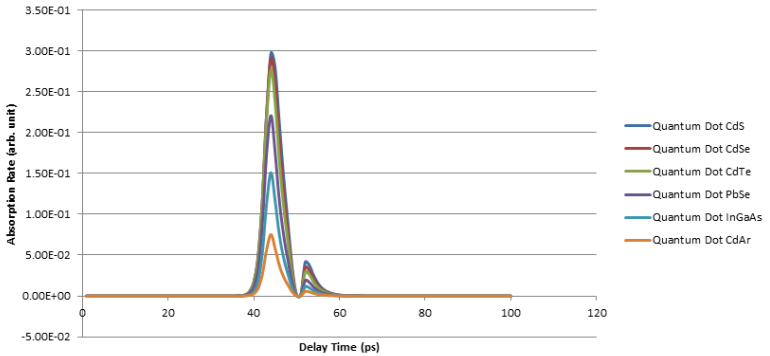
```
In [546]: ada = tensor(num(N), qeye(3))
          c_ops = []  # Build collapse operators
          E_0 = 50
          f = 1000
          f2 = 1000
          kappa = 0.5 # Cavity decay rate
          c_ops.append(np.sqrt(kappa) * a)
          gamma = 9  # Decay Rate
          c_ops.append(np.sqrt(4*gamma/13) * sigma_ue) # Menentukan branching ratio, sebesar 4/13 dari state eksitasi ke steady
          c_ops.append(np.sqrt(9*gamma/13) * sigma_ge) # 9/13 dari state ground ->eksitasi
          times = np.linspace(0, 150, 100) # #waktu total
          psi0 = tensor(basis(N, 0), ustate) # State awal sistem
          state_GG = tensor(basis(N, 1), ground) # State yang diproyeksikan, yakni ground state (GG) dan steady state (UU)
          sigma_GG = state_GG * state_GG.dag()
          state_UU = tensor(basis(N, 0), ustate)
          sigma_UU = state_UU * state_UU.dag()
          g = 5  # konstanta coupling
          H0 = -g * (sigma_ge.dag() * a + a.dag() * sigma_ge)  # Hamiltonian tanpa waktu
          H1 = f - (sigma_ge.dag() + sigma_ge)  # Hamiltonian dengan waktu #1
          H2 = f2 - (sigma_ue.dag() + sigma_ue)  # Hamiltonian dengan waktu #2
          times2 = np.linspace(75, 150, 100) #Waktu yang digunakan laser untuk perpindahan ground -> excited
          times3 = np.linspace(75,100, 100) #Waktu yang digunakan laser untuk perpindahan excited -> steady state (u)
          options.nsteps = 1000000 #mendefinisikan step integrasi mesolve sebanyak satu juta langkah

In [582]: def H1_coeff(t2, t3):
              sigma = 8
              ta = 75
              I_a = E_0**2 / (np.sqrt(np.pi)*sigma) * np.exp(-(t3 - ta)**2/sigma**2) * np.exp(-(t3-t2)**2 / (4*sigma**2))
              E_tot = I_a
              return E_tot #Fungsi matematis laser pump

In [583]: def H2_coeff(t, t3):
              sigma2 = 8
              tb = 60
              I_b = E_0**2 / (np.sqrt(np.pi)*sigma2) * np.exp(-(t - tb)**2/sigma2**2) * np.exp(-(t-t3)**2 / (4*sigma2**2))
              E_tot = I_b
              return E_tot #Fungsi matematis laser probe
```

Gambar 4.5 Pendefinisian matriks proyektor, Hamiltonian pada laser *pump* dan *probe*, serta operator *collapse*.

Perlu diketahui bahwa satuan waktu yang dipakai pada sistem ini diasumsikan pada skala pikosekon.

4.2.2 Hasil dan Analisis Data Simulator

Setelah menjalankan fungsi mesolve() pada simulator, grafik dari hasil simulator dapat dilihat pada gambar 4.6 dengan cara menggunakan library matplotlib keperluan penggambaran grafik..



Gambar 4.6 Output dari simulator *pump-probe spectroscopy* dari berbagai jenis *quantum dot*

Data pada gambar 4.6 menunjukkan data yang dihasilkan dari sistem *pump-probe spectroscopy* pada suatu sample *band gap energy* yang berbeda-beda, yakni larutan *quantum dot* CdS, CdSe, CdTe, PbSe, InGaAs, dan CdAr. Terlihat bahwa perubahan nilai delay mengubah grafik eksitasi dari zat tersebut, sebagaimana dari hasil eksperimen yang ditunjukkan pada gambar 4.2 dengan penambahan nilai absorbansi/*expectation value* dari terhadap perubahan *band gap energy*. Hal ini membuktikan bahwa simulator sistem *pump-probe spectroscopy* yang dibuat terverifikasi oleh hasil eksperimen sebelumnya.

4.3 Pengujian *Classifier*/Pengklasifikiasi

4.3.1 Perancangan *Recurrent Neural Network*

Tahap terakhir yang dilakukan dalam perancangan sistem *pump-probe spectroscopy* ialah mendesain pengklasifikasi data sistem dengan menggunakan metode pembelajaran mesin/*machine learning*. Teknik *machine learning* yang digunakan pada desain sistem ini ialah *recurrent neural network* yang ditujukan untuk mengklasifikasi data sekuensial seperti data yang berubah terhadap waktu.

Pemanggilan library dan penentuan *random seed* yang dipakai terlihat pada gambar 4.9. Penentuan *random seed* ditujukan untuk mengatur keacakan/*randomness* dari metode *machine learning* yang

dipakai agar hasil yang didapatkan tidak terlalu jauh dari target yang diinginkan.

```
In [1]: import numpy as np
        import tensorflow as tf
        from tensorflow import keras
        import pandas as pd
        import seaborn as sns
        from pylab import rcParams
        import matplotlib.pyplot as plt
        from matplotlib import rc
        from pandas.plotting import register_matplotlib_converters
        from sklearn.preprocessing import RobustScaler
        %matplotlib inline
        %config InlineBackend.figure_format='retina'

        register_matplotlib_converters()
        sns.set(style='whitegrid', palette='muted', font_scale=1.5)

        rcParams['figure.figsize'] = 22, 10

        RANDOM_SEED = 42

        np.random.seed(RANDOM_SEED)
        tf.random.set_seed(RANDOM_SEED)

In [2]: column_names = ['Target', 'Y', 'Number']

        #df = pd.read_csv('C:\\Users\\Admin\\Documents\\xshkx-m78es.csv', header=None, skiprows=[0], names=column_names)
        df = pd.read_csv('C:\\Users\\Admin\\Documents\\9e5my-obiow.csv', header=None, skiprows=[0], names=column_names)
        #df_train = df
        df_train = df[df['Number'] <= 23405]
        df_test = df[df['Number'] > 23405]

        #df_test = pd.read_csv('C:\\Users\\Admin\\Documents\\svxcj-jssz9.csv', header=None, skiprows=[0], names=column_names)

        #df_test.head
```

Gambar 4.7 Pemanggilan *library*, penentuan *random seed*, dan pemanggilan data.

Adapun pada gambar 4.7 juga dataset yang hendak digunakan dipanggil terlebih dahulu menggunakan library Pandas, dan kemudian data dibagi menjadi dua, yakni data *training* dan data *test*. Dataset tersebut pun kemudian diberikan label berupa “Target”, “Y”, dan “Number”, yang masing-masing ditujukan untuk menentukan jenis larutan (0 untuk larutan *carbon dot* lengkuas dan 1 untuk larutan *carbon dot* jahe), nilai absorbansi, serta pelabelan sumbu x dari angka 1 sampai 30954. Pembagian data *training* dan *testing* dilakukan dengan membagi dua nilai total sumbu x tersebut, sehingga menjadi 15477 data. Data di bawah nilai 15477 dijadikan data *training* dan diatas angka tersebut dijadikan data *testing*.

```
In [5]: scale_columns = ['Y']

        scaler = RobustScaler()

        scaler = scaler.fit(df_train[scale_columns])

        df_train.loc[:, scale_columns] = scaler.transform(df_train[scale_columns].to_numpy())
        df_test.loc[:, scale_columns] = scaler.transform(df_test[scale_columns].to_numpy())
```

Gambar 4.8 Skala ulang data dengan cara mendefinisikan variabel *scaler* dan konversi data menjadi array numpy.

Data yang telah diskala ulang sesuai pada gambar 4.8 kemudian transformasi agar sesuai dengan yang dibutuhkan *library* numpy. Hal ini dilakukan guna memudahkan proses preprocessing yang akan dijelaskan selanjutnya.

4.3.2 Proses *Preprocessing* dan Pengolahan Dataset

Proses *preprocessing* dari pengklasifikasi data sample dapat dilihat pada gambar 4.9. Dataset kemudian dikonversi menjadi array numpy. Pada gambar 4.10 telah ditunjukkan pemasangan label pada data training dengan fungsi encoder.

```
In [6]: from scipy import stats

        def create_dataset(X, y, time_steps=1, step=1):
            Xs, ys = [], []
            for i in range(0, len(X) - time_steps, step):
                v = X.iloc[i:(i + time_steps)].values
                labels = y.iloc[i: i + time_steps]
                Xs.append(v)
                ys.append(stats.mode(labels)[0][0])
            return np.array(Xs), np.array(ys).reshape(-1, 1)

        TIME_STEPS = 200
        STEP = 40

        X_train, y_train = create_dataset(
            df_train[['Y']],
            df_train.Target,
            TIME_STEPS,
            STEP
        )

        X_test, y_test = create_dataset(
            df_test[['Y']],
            df_test.Target,
            TIME_STEPS,
            STEP
        )
```

Gambar 4.9 proses *preprocessing* pertama

```
In [8]: from sklearn.preprocessing import OneHotEncoder

        enc = OneHotEncoder(handle_unknown='ignore', sparse=False)

        enc = enc.fit(y_train)

        y_train = enc.transform(y_train)
        y_test = enc.transform(y_test)
```

Gambar 4.10 Pemasangan *encoder* pada data training.

Setelah itu data training dimasukkan pada layer LSTM yang didefinisikan untuk variabel *model*. Adapun dalam model ini digunakan juga fungsi aktivasi *softmax* dan ReLU, kemudian digunakan fungsi compile untuk memodelkan data loss sebagaimana ditunjukkan pada gambar 4.11

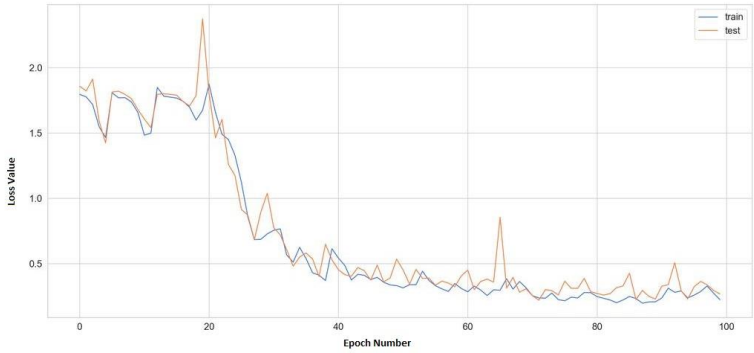
```
In [10]: model = keras.Sequential()
         model.add(
             keras.layers.Bidirectional(
               keras.layers.LSTM(
                   units=128,
                   input_shape=[X_train.shape[1], X_train.shape[2]]
               )
             )
         )
         model.add(keras.layers.Dropout(rate=0.5))
         model.add(keras.layers.Dense(units=128, activation='relu'))
         model.add(keras.layers.Dense(y_train.shape[1], activation='softmax'))
         model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
```

```
In [11]: history = model.fit(
             X_train, y_train,
             epochs=1200,
             batch_size=64,
             validation_split=0.1,
             shuffle=True
         )
```

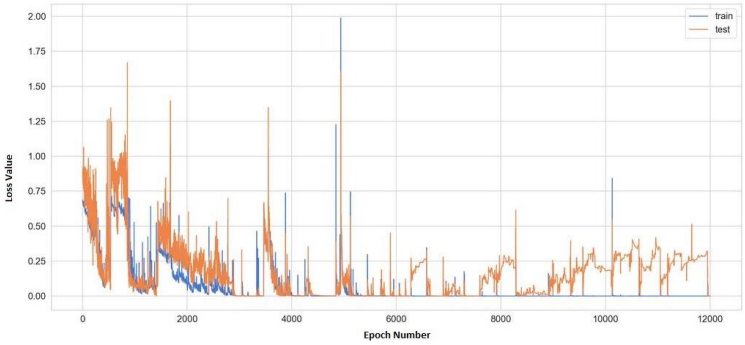
Gambar 4.11 Proses pembuatan model dan training

4.3.3 Hasil Klasifikasi dan Analisa

Training sebanyak 12000 epoch telah dilakukan, sesuai pada gambar 4.12 dan 4.13. Dapat dilihat bahwa pada waktu tertentu pasti terjadi lonjakan pada nilai loss dan *accuracy* pada data, namun data tersebut cenderung berkurang semakin tingginya epoch yang digunakan dengan nilai loss yang dihasilkan mendekati 0%.



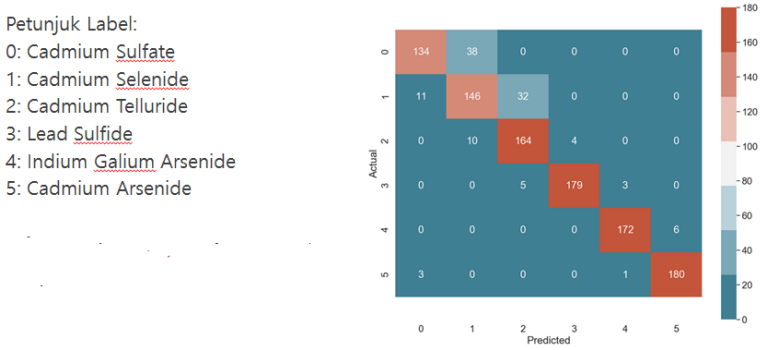
Gambar 4.12 Grafik akurasi model dan *loss* pada dataset *quantum dot* terhadap jumlah epoch sebanyak 100



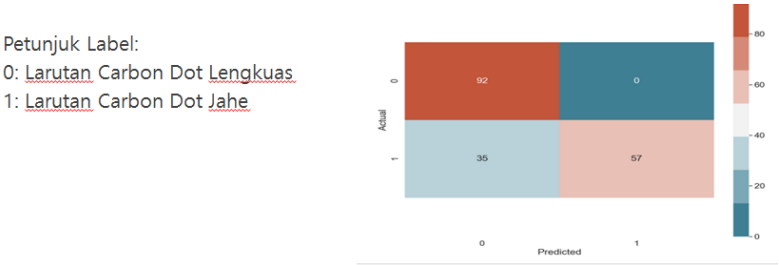
Gambar 4.13 Grafik akurasi model dan *loss* pada dataset *carbon quantum dot* terhadap jumlah epoch sebanyak 12000.

Akurasi dari model yang telah dilatih kemudian dimodelkan melalui confusion matrix yang dapat dilihat pada gambar 4.14 dan 4.15. Pada *confusion* matrix di gambar 4.14, label yang diberikan dari angka 0 sampai 5 masing-masing merupakan *quantum dot* jenis CdS, CdSe, CdTe, PbSe, InGaAs, dan CdAr, sedangkan pada gambar 4.15, 0 menandakan larutan *carbon quantum dot* lengkuas dan 1 adalah larutan *carbon quantum dot* jahe. Terlihat jelas bahwa model memiliki akurasi yang cukup tinggi, yakni 90% untuk prediksi data dari keenam jenis

quantum dot dan 81% untuk prediksi data larutan *carbon dot* lengkuas dibandingkan jahe. Akurasi pada permodelan untuk larutan *carbon quantum dot* terjadi karena adanya noise yang terdapat pada pengambilan data sample, sehingga tidak ada satu data yang sama dengan yang lainnya pada satu jenis sample yang sama.



Gambar 4.14 *Confusion matrix* pada data untuk berbagai jenis quantum dot.



Gambar 4.15 *Confusion matrix* pada data untuk berbagai jenis larutan *carbon dot*

BAB 5
KESIMPULAN & SARAN

5.1 KESIMPULAN

Dari sistem *pump probe* yang telah dibuat di atas, dapat disimpulkan bahwa sistem *pump-probe spectroscopy* yang telah disusun memiliki kemampuan dengan potensial yang sangat tinggi dalam memodelkan dinamika elektron pada suatu bahan. Pada hasil eksperimen didapatkan nilai absorbansi terhadap perubahan waktu delay pada dua jenis sample dengan nilai puncak sekitar 0.35 satuan absorbansi. Hal tersebut digunakan untuk membuktikan bahwa teori serta eksperimen dari sistem *pump-probe spectroscopy* berjalan selaras, dengan hasil dari komputasi simulator mengalami fenomena yang sama dengan hasil yang didapatkan pada sample dengan setup eksperimen, yakni terdapat pergeseran grafik dari waktu delay 40 sekon ke 60 sekon dan penambahan nilai satuan absorbansi dari 0.01 ke 0.25. Selain itu, program pengklasifikasi data sample yang menggunakan metode RNN dapat menghasilkan model dengan akurasi yang sangat tinggi pada angka 90% pada pelatihan 100 epoch untuk larutan *quantum dot*, dengan angka loss terakhir tercatat di bawah 2% dan 81% untuk larutan *carbon dot* pada pelatihan 12000 epoch.

5.2 SARAN

Perlu ditambahkan variabel lain selain *band gap energy* suatu zat untuk dapat memodelkan sistem *pump-probe spectroscopy* pada zat yang lebih kompleks, serta perbanyakan sample pada setup eksperimen untuk menambahkan data yang dapat digunakan untuk melakukan verifikasi pada sistem simulator. Selain itu, pengklasifikasi dapat dibuat dengan menggunakan metode lain yang memungkinkan penambahan akurasi pada model yang dilatih.

Halaman ini sengaja dikosongkan.

DAFTAR PUSAKA

[1] M. Fushitani, “Applications of pump-probe spectroscopy,” Applications of pump-probe spectroscopy, no. 104, pp. 272–297, May 2008.

[2] D. J. Griffiths, Introduction to Electrodynamics. Cambridge, United Kingdom: Cambridge University Press, 2018.

[3] C. B. Hitz, J. J. Ewing, and J. Hecht, Introduction to laser technology. Hoboken, NJ: Wiley-IEEE Press, 2012.

[4] R. Paschotta, “Beam Splitters,” RP Photonics Encyclopedia - beam splitters, optical ower splitter, beamsplitter, thin-film polarizer, non-polarizing beam splitter cubes, important properties, 23-Oct-2019.[Online].:https://www.rpphotonics.com/beam_splitters.html. [Accessed: 01-Nov-2019].

[5] N. Zettili, Quantum Mechanics: Concepts and Applications, 2nd Edition. John Wiley & Sons, 2009.

[6] Z. Li, J. Fang and C. Martens, "Simulation of ultrafast dynamics and pump–probe spectroscopy using classical trajectories", The Journal of Chemical Physics, vol. 104, no. 18, pp. 6919-6929, 1996. Available: 10.1063/1.471407 [Accessed 8 June 2020].

[7] D. Manzano, "A short introduction to the Lindblad master equation", AIP Advances, vol. 10, no. 2, p. 025106, 2020. Available: 10.1063/1.5115323 [Accessed 8 June 2020].

[8] Functions — QuTiP 4.0 Documentation", Qutip.org. [Online]. Available: http://qutip.org/docs/4.0.2/apidoc/functions.html?highlight=mesolve#module-qutip.mesolve. [Accessed: 08- Jun- 2020].

[9] ”An Introduction to the Technique and Applications of Pump-Probe Spectroscopy,” Special Topic Paper, Physics 211A, pp. 1–7, 2014.

[10] P.-T. Dong and J.-X. Cheng, “Pump–Probe Microscopy: Theory, Instrumentation, and Applications,” Spectroscopy, vol. 32, no. 4, pp. 1–9, Apr. 2017.

[11] S. A. Miller, “Systems and Methods for Pump-Probe Spectroscopy”, US 2019/0056313 A1, 2019.

[12] J. Curtis, T. Tokumoto, N. Nolan, L. Mcclintock, J. Cherian, S. Mcgill, and D. J. Hilton, "Ultrafast Pump-probe Spectroscopy in Gallium Arsenide at 25 Tesla," Cleo: 2015, 2015.

[13] K. J. Nordell, E. M. Boatman, and G. C. Lisensky, “A Safer, Easier, Faster Synthesis for CdSe Quantum Dot Nanocrystals,” Journal of Chemical Education, vol. 82, no. 11, p. 1697, 2005.

[14] E. O. Chukwuocha, M. C. Onyeaju, and T. S. T. Harry, “Theoretical Studies on the Effect of Confinement on Quantum Dots Using the Brus Equation,” World Journal of Condensed Matter Physics, vol. 02, no. 02, pp. 96–100, 2012.
[15] "What is LabVIEW? - National Instruments", Ni.com, 2020. [Online]. Available: https://www.ni.com/en-id/shop/labview.html. [Accessed: 08- Jun- 2020].
[16] A. Y. Deviatov, I. A. Iakovlev, and V. V. Mazurenko, “Recurrent network classifier for ultrafast skyrmion dynamics,” Jul. 2019.
[17] Z. C. Lipton, J. Berkowitz, and C. Elkan, “A Critical Review of Recurrent Neural Networks for Sequence Learning,” arXiv:1506.00019, Jun. 2015.
[18] B. Prijono, "Pengenalan Recurrent Neural Network (RNN) – Bagian 1", Belajar Pembelajaran Mesin Indonesia, 2018. [Online]. Available: https://indoml.com/2018/04/04/pengenalan-rnn-bag-1/. [Accessed: 08- Jun- 2020].
[19] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory", Neural Computation, vol. 9, no. 8, pp. 1735-1780, 1997. Available: 10.1162/neco.1997.9.8.1735 [Accessed 8 June 2020].
[20] B. Prijono, "Pengenalan Recurrent Neural Network (RNN) – Bagian 1", Belajar Pembelajaran Mesin Indonesia, 2018. [Online]. Available: https://indoml.com/2018/04/04/pengenalan-rnn-bag-1/. [Accessed: 08- Jun- 2020].
[21] Johansson, P. Nation and F. Nori, "QuTiP: An open-source Python framework for the dynamics of open quantum systems", Computer Physics Communications, vol. 183, no. 8, pp. 1760-1772, 2012. Available: 10.1016/j.cpc.2012.02.021 [Accessed 8 June 2020].
[22] "TensorFlow", *TensorFlow*, 2020. [Online]. Available: https://www.tensorflow.org/. [Accessed: 14- Jul- 2020].
[23] Ocean Insights, “Maya2000 Pro SpectrometerHigh Sensitivity and Deep UV Measurement,” Maya2000 Pro datasheet, 2013.
[24] Newport, “DL Series High-Performance Delay Line Stages”, DL125 datasheet, 2014.

LAMPIRAN

1) Dokumentasi Setup Eksperimen

Berikut ialah dokumentasi dari situasi dalam perancangan setup eksperimen untuk sistem *pump-probe spectroscopy*.

Laser Femtosekon

Setup Eksperimen

2) Kode program untuk fungsi mesolve() pada QuTiP

```
__all__ = ['mesolve', 'odesolve']

import os
import types
from functools import partial
import numpy as np
import scipy.sparse as sp
import scipy.integrate
import warnings

from qutip.qobj import Qobj, isket, isoper, issuper
from qutip.superoperator import spre, spost, liouvillian, mat2vec,
vec2mat
from qutip.expect import expect_rho_vec
from qutip.solver import Options, Result, config, _solver_safety_check
from qutip.cy.spmatfuncs import cy_ode_rhs, cy_ode_rho_func_td
from qutip.cy.spconvert import dense2D_to_fastcsr_fmode
from qutip.cy.codegen import Codegen
from qutip.cy.utilities import _cython_build_cleanup
from qutip.rhs_generate import rhs_generate
from qutip.states import ket2dm
from qutip.rhs_generate import _td_format_check, _td_wrap_array_str
from qutip.interpolate import Cubic_Spline
from qutip.settings import debug

from qutip.sesolve import (_sesolve_list_func_td, _sesolve_list_str_td,
                           _sesolve_list_td, _sesolve_func_td, _sesolve_const)

from qutip.ui.progressbar import BaseProgressBar, TextProgressBar

if debug:
    import inspect


# -----------------------------------------------------------------------------
```

```
# pass on to wavefunction solver or master equation solver depending on whether
# any collapse operators were given.
#
[docs]
def mesolve(H, rho0, tlist, c_ops=[], e_ops=[], args={}, options=None,
            progress_bar=None, _safe_mode=True):
    """
    Master equation evolution of a density matrix for a given Hamiltonian and
    set of collapse operators, or a Liouvillian.

    Evolve the state vector or density matrix (`rho0`) using a given
    Hamiltonian (`H`) and an [optional] set of collapse operators
    (`c_ops`), by integrating the set of ordinary differential equations
    that define the system. In the absence of collapse operators the system is
    evolved according to the unitary evolution of the Hamiltonian.

    The output is either the state vector at arbitrary points in time
    (`tlist`), or the expectation values of the supplied operators
    (`e_ops`). If e_ops is a callback function, it is invoked for each
    time in `tlist` with time and the state as arguments, and the function
    does not use any return values.

    If either `H` or the Qobj elements in `c_ops` are superoperators, they
    will be treated as direct contributions to the total system Liouvillian.
    This allows to solve master equations that are not on standard Lindblad
    form by passing a custom Liouvillian in place of either the `H` or `c_ops`
    elements.

    **Time-dependent operators**

    For time-dependent problems, `H` and `c_ops` can be callback
    functions that takes two arguments, time and `args`, and returns the
    Hamiltonian or Liouvillian for the system at that point in time
    (*callback format*).
```

Alternatively, `H` and `c_ops` can be a specified in a nested-list format
where each element in the list is a list of length 2, containing an operator (:class:`qutip.qobj`) at the first element and where the second element is either a string (*list string format*), a callback function (*list callback format*) that evaluates to the time-dependent coefficient for the corresponding operator, or a NumPy array (*list array format*) which specifies the value of the coefficient to the corresponding operator for each value of t in tlist.

Examples

H = [[H0, 'sin(w*t)'], [H1, 'sin(2*w*t)']]

H = [[H0, f0_t], [H1, f1_t]]

where f0_t and f1_t are python functions with signature f_t(t, args).

H = [[H0, np.sin(w*tlist)], [H1, np.sin(2*w*tlist)]]

In the *list string format* and *list callback format*, the string expression and the callback function must evaluate to a real or complex
number (coefficient for the corresponding operator).

In all cases of time-dependent operators, `args` is a dictionary of parameters that is used when evaluating operators. It is passed to the callback functions as second argument.

Additional options

Additional options to mesolve can be set via the `options` argument, which
should be an instance of :class:`qutip.solver.Options`. Many ODE integration options can be set this way, and the `store_states` and `store_final_state` options can be used to store states even though expectation values are requested via the `e_ops` argument.

.. note::

If an element in the list-specification of the Hamiltonian or the list of collapse operators are in superoperator form it will be added to the total Liouvillian of the problem with out further transformation. This allows for using mesolve for solving master equations that are not on standard Lindblad form.

.. note::

On using callback function: mesolve transforms all :class:`qutip.qobj`
objects to sparse matrices before handing the problem to the integrator
function. In order for your callback function to work correctly, pass all :class:`qutip.qobj` objects that are used in constructing the Hamiltonian via args. mesolve will check for :class:`qutip.qobj` in `args` and handle the conversion to sparse matrices. All other :class:`qutip.qobj` objects that are not passed via `args` will be passed on to the integrator in scipy which will raise an NotImplemented
exception.

Parameters

H : :class:`qutip.Qobj`
System Hamiltonian, or a callback function for time-dependent Hamiltonians, or alternatively a system Liouvillian.

rho0 : :class:`qutip.Qobj`
initial density matrix or state vector (ket).

tlist : *list* / *array*
list of times for :math:`t`.

c_ops : list of :class:`qutip.Qobj`
single collapse operator, or list of collapse operators, or a list of Liouvillian superoperators.

```
    e_ops : list of :class:`qutip.Qobj` / callback function single
        single operator or list of operators for which to evaluate
        expectation values.

    args : *dictionary*
        dictionary of parameters for time-dependent Hamiltonians and
        collapse operators.

    options : :class:`qutip.Options`
        with options for the solver.

    progress_bar: BaseProgressBar
        Optional instance of BaseProgressBar, or a subclass thereof, for
        showing the progress of the simulation.

    Returns
    -------

    result: :class:`qutip.Result`

        An instance of the class :class:`qutip.Result`, which contains
        either an *array* `result.expect` of expectation values for the times
        specified by `tlist`, or an *array* `result.states` of state vectors or
        density matrices corresponding to the times in `tlist` [if `e_ops` is
        an empty list], or nothing if a callback function was given in place
of
        operators for which to calculate the expectation values.

    """

    if _safe_mode:
        _solver_safety_check(H, rho0, c_ops, e_ops, args)


    if progress_bar is None:
        progress_bar = BaseProgressBar()
    elif progress_bar is True:
        progress_bar = TextProgressBar()
```

```
    # check whether c_ops or e_ops is is a single operator
    # if so convert it to a list containing only that operator
    if isinstance(c_ops, Qobj):
        c_ops = [c_ops]

    if isinstance(e_ops, Qobj):
        e_ops = [e_ops]

    if isinstance(e_ops, dict):
        e_ops_dict = e_ops
        e_ops = [e for e in e_ops.values()]
    else:
        e_ops_dict = None

    # check if rho0 is a superoperator, in which case e_ops argument
should
    # be empty, i.e., e_ops = []
    if issuper(rho0) and not e_ops == []:
        raise TypeError("Must have e_ops = [] when initial condition rho0
is" +
                        " a superoperator.")

    # convert array based time-dependence to string format
    H, c_ops, args = _td_wrap_array_str(H, c_ops, args, tlist)

    # check for type (if any) of time-dependent inputs
    _, n_func, n_str = _td_format_check(H, c_ops)

    if options is None:
        options = Options()

    if (not options.rhs_reuse) or (not config.tdfunc):
        # reset config collapse and time-dependence flags to default values
        config.reset()

    res = None

    #
```

```
# dispatch the appropriate solver
#
if ((c_ops and len(c_ops) > 0)
   or (not isket(rho0))
   or (isinstance(H, Qobj) and issuper(H))
   or (isinstance(H, list) and
      isinstance(H[0], Qobj) and issuper(H[0]))):

   #
   # we have collapse operators, or rho0 is not a ket,
   # or H is a Liouvillian
   #

   #
   # find out if we are dealing with all-constant hamiltonian and
   # collapse operators or if we have at least one time-dependent
   # operator. Then delegate to appropriate solver...
   #

   if isinstance(H, Qobj):
      # constant hamiltonian
      if n_func == 0 and n_str == 0:
         # constant collapse operators
         res = _mesolve_const(H, rho0, tlist, c_ops,
                              e_ops, args, options,
                              progress_bar)
      elif n_str > 0:
         # constant hamiltonian but time-dependent collapse
         # operators in list string format
         res = _mesolve_list_str_td([H], rho0, tlist, c_ops,
                                    e_ops, args, options,
                                    progress_bar)
      elif n_func > 0:
         # constant hamiltonian but time-dependent collapse
         # operators in list function format
         res = _mesolve_list_func_td([H], rho0, tlist, c_ops,
                                     e_ops, args, options,
                                     progress_bar)
```

```
    elif isinstance(H, (types.FunctionType,
                        types.BuiltinFunctionType, partial)):
        # function-callback style time-dependence: must have constant
        # collapse operators
        if n_str > 0:  # or n_func > 0:
            raise TypeError("Incorrect format: function-format " +
                            "Hamiltonian cannot be mixed with " +
                            "time-dependent collapse operators.")
        else:
            res = _mesolve_func_td(H, rho0, tlist, c_ops,
                                   e_ops, args, options,
                                   progress_bar)

    elif isinstance(H, list):
        # determine if we are dealing with list of [Qobj, string] or
        # [Qobj, function] style time-dependencies (for pure python and
        # cython, respectively)
        if n_func > 0:
            res = _mesolve_list_func_td(H, rho0, tlist, c_ops,
                                        e_ops, args, options,
                                        progress_bar)
        else:
            res = _mesolve_list_str_td(H, rho0, tlist, c_ops,
                                       e_ops, args, options,
                                       progress_bar)

    else:
        raise TypeError("Incorrect specification of Hamiltonian " +
                        "or collapse operators.")

else:
    #
    # no collapse operators: unitary dynamics
    #
    if n_func > 0:
        res = _sesolve_list_func_td(H, rho0, tlist,
                                    e_ops, args, options, progress_bar)
    elif n_str > 0:
        res = _sesolve_list_str_td(H, rho0, tlist,
```

```
                                e_ops, args, options, progress_bar)
        elif isinstance(H, (types.FunctionType,
                            types.BuiltinFunctionType, partial)):
            res = _sesolve_func_td(H, rho0, tlist,
                                   e_ops, args, options, progress_bar)
        else:
            res = _sesolve_const(H, rho0, tlist,
                                 e_ops, args, options, progress_bar)

    if e_ops_dict:
        res.expect = {e: res.expect[n]
                      for n, e in enumerate(e_ops_dict.keys())}

    return res



# -----------------------------------------------------------------------------
# A time-dependent dissipative master equation on the list-function format
#
def _mesolve_list_func_td(H_list, rho0, tlist, c_list, e_ops, args, opt,
                          progress_bar):
    """
    Internal function for solving the master equation. See mesolve for usage.
    """

    if debug:
        print(inspect.stack()[0][3])

    #
    # check initial state
    #
    if isket(rho0):
        rho0 = rho0 * rho0.dag()

    #
    # construct liouvillian in list-function format
```

```
#
L_list = []
if opt.rhs_with_state:
    constant_func = lambda x, y, z: 1.0
else:
    constant_func = lambda x, y: 1.0

# add all hamitonian terms to the lagrangian list
for h_spec in H_list:

    if isinstance(h_spec, Qobj):
        h = h_spec
        h_coeff = constant_func

    elif isinstance(h_spec, list) and isinstance(h_spec[0], Qobj):
        h = h_spec[0]
        h_coeff = h_spec[1]

    else:
        raise TypeError("Incorrect specification of time-dependent " +
                        "Hamiltonian (expected callback function)")

    if isoper(h):
        L_list.append([(-1j * (spre(h) - spost(h))).data, h_coeff, False])

    elif issuper(h):
        L_list.append([h.data, h_coeff, False])

    else:
        raise TypeError("Incorrect specification of time-dependent " +
                        "Hamiltonian (expected operator or superoperator)")

# add all collapse operators to the liouvillian list
for c_spec in c_list:

    if isinstance(c_spec, Qobj):
        c = c_spec
        c_coeff = constant_func
        c_square = False
```

```
    elif isinstance(c_spec, list) and isinstance(c_spec[0], Qobj):
        c = c_spec[0]
        c_coeff = c_spec[1]
        c_square = True

    else:
        raise TypeError("Incorrect specification of time-dependent " +
                        "collapse operators (expected callback function)")

    if isoper(c):
        L_list.append([liouvillian(None, [c], data_only=True),
                       c_coeff, c_square])

    elif issuper(c):
        L_list.append([c.data, c_coeff, c_square])

    else:
        raise TypeError("Incorrect specification of time-dependent " +
                        "collapse operators (expected operator or " +
                        "superoperator)")

#
# setup integrator
#
initial_vector = mat2vec(rho0.full()).ravel('F')
if issuper(rho0):
    if opt.rhs_with_state:
        r = scipy.integrate.ode(dsuper_list_td_with_state)
    else:
        r = scipy.integrate.ode(dsuper_list_td)
else:
    if opt.rhs_with_state:
        r = scipy.integrate.ode(drho_list_td_with_state)
    else:
        r = scipy.integrate.ode(drho_list_td)
r.set_integrator('zvode', method=opt.method, order=opt.order,
                 atol=opt.atol, rtol=opt.rtol, nsteps=opt.nsteps,
                 first_step=opt.first_step, min_step=opt.min_step,
```

```
                    max_step=opt.max_step)
    r.set_initial_value(initial_vector, tlist[0])
    r.set_f_params(L_list, args)

    #
    # call generic ODE code
    #
    return _generic_ode_solve(r, rho0, tlist, e_ops, opt, progress_bar)


#
# evaluate drho(t)/dt according to the master equation using the
# [Qobj, function] style time dependence API
#
def drho_list_td(t, rho, L_list, args):

    L = L_list[0][0] * L_list[0][1](t, args)
    for n in range(1, len(L_list)):
        #
        # L_args[n][0] = the sparse data for a Qobj in super-operator form
        # L_args[n][1] = function callback giving the coefficient
        #
        if L_list[n][2]:
            L = L + L_list[n][0] * (L_list[n][1](t, args)) ** 2
        else:
            L = L + L_list[n][0] * L_list[n][1](t, args)

    return L * rho


def drho_list_td_with_state(t, rho, L_list, args):

    L = L_list[0][0] * L_list[0][1](t, rho, args)
    for n in range(1, len(L_list)):
        #
        # L_args[n][0] = the sparse data for a Qobj in super-operator form
        # L_args[n][1] = function callback giving the coefficient
        #
        if L_list[n][2]:
```

```
            L = L + L_list[n][0] * (L_list[n][1](t, rho, args)) ** 2
        else:
            L = L + L_list[n][0] * L_list[n][1](t, rho, args)

    return L * rho

#
# evaluate dE(t)/dt according to the master equation using the
# [Qobj, function] style time dependence API, where E is a superoperator
#
def dsuper_list_td(t, y, L_list, args):

    L = L_list[0][0] * L_list[0][1](t, args)
    for n in range(1, len(L_list)):
        #
        # L_args[n][0] = the sparse data for a Qobj in super-operator form
        # L_args[n][1] = function callback giving the coefficient
        #
        if L_list[n][2]:
            L = L + L_list[n][0] * (L_list[n][1](t, args)) ** 2
        else:
            L = L + L_list[n][0] * L_list[n][1](t, args)

    return _ode_super_func(t, y, L)

def dsuper_list_td_with_state(t, y, L_list, args):

    L = L_list[0][0] * L_list[0][1](t, y, args)
    for n in range(1, len(L_list)):
        #
        # L_args[n][0] = the sparse data for a Qobj in super-operator form
        # L_args[n][1] = function callback giving the coefficient
        #
        if L_list[n][2]:
            L = L + L_list[n][0] * (L_list[n][1](t, y, args)) ** 2
        else:
            L = L + L_list[n][0] * L_list[n][1](t, y, args)
```

```
    return _ode_super_func(t, y, L)

# -----------------------------------------------------------------------
# A time-dependent dissipative master equation on the list-string format
for
# cython compilation
#
def _mesolve_list_str_td(H_list, rho0, tlist, c_list, e_ops, args, opt,
                         progress_bar):
    """
    Internal function for solving the master equation. See mesolve for
usage.
    """

    if debug:
        print(inspect.stack()[0][3])

    #
    # check initial state: must be a density matrix
    #
    if isket(rho0):
        rho0 = rho0 * rho0.dag()

    #
    # construct liouvillian
    #
    Lconst = 0

    Ldata = []
    Linds = []
    Lptrs = []
    Lcoeff = []
    Lobj = []

    # loop over all hamiltonian terms, convert to superoperator form and
    # add the data of sparse matrix representation to
    for h_spec in H_list:

        if isinstance(h_spec, Qobj):
```

```
            h = h_spec

            if isoper(h):
                Lconst += -1j * (spre(h) - spost(h))
            elif issuper(h):
                Lconst += h
            else:
                raise TypeError("Incorrect specification of time-dependent " +
                                "Hamiltonian (expected operator or " +
                                "superoperator)")

        elif isinstance(h_spec, list):
            h = h_spec[0]
            h_coeff = h_spec[1]

            if isoper(h):
                L = -1j * (spre(h) - spost(h))
            elif issuper(h):
                L = h
            else:
                raise TypeError("Incorrect specification of time-dependent " +
                                "Hamiltonian (expected operator or " +
                                "superoperator)")

            Ldata.append(L.data.data)
            Linds.append(L.data.indices)
            Lptrs.append(L.data.indptr)
            if isinstance(h_coeff, Cubic_Spline):
                Lobj.append(h_coeff.coeffs)
            Lcoeff.append(h_coeff)

        else:
            raise TypeError("Incorrect specification of time-dependent " +
                            "Hamiltonian (expected string format)")

    # loop over all collapse operators
    for c_spec in c_list:

        if isinstance(c_spec, Qobj):
```

```
            c = c_spec

            if isoper(c):
                cdc = c.dag() * c
                Lconst += spre(c) * spost(c.dag()) - 0.5 * spre(cdc) \
                                        - 0.5 * spost(cdc)
            elif issuper(c):
                Lconst += c
            else:
                raise TypeError("Incorrect specification of time-dependent " +
                            "Liouvillian (expected operator or " +
                            "superoperator)")

        elif isinstance(c_spec, list):
            c = c_spec[0]
            c_coeff = c_spec[1]

            if isoper(c):
                cdc = c.dag() * c
                L = spre(c) * spost(c.dag()) - 0.5 * spre(cdc) \
                                    - 0.5 * spost(cdc)
                c_coeff = "(" + c_coeff + ")**2"
            elif issuper(c):
                L = c
            else:
                raise TypeError("Incorrect specification of time-dependent " +
                            "Liouvillian (expected operator or " +
                            "superoperator)")

            Ldata.append(L.data.data)
            Linds.append(L.data.indices)
            Lptrs.append(L.data.indptr)
            Lcoeff.append(c_coeff)

        else:
            raise TypeError("Incorrect specification of time-dependent " +
                        "collapse operators (expected string format)")

    # add the constant part of the lagrangian
```

```
if Lconst != 0:
    Ldata.append(Lconst.data.data)
    Linds.append(Lconst.data.indices)
    Lptrs.append(Lconst.data.indptr)
    Lcoeff.append("1.0")

# the total number of liouvillian terms (hamiltonian terms +
# collapse operators)
n_L_terms = len(Ldata)

#
# setup ode args string: we expand the list Ldata, Linds and Lptrs into
# and explicit list of parameters
#
string_list = []
for k in range(n_L_terms):
    string_list.append("Ldata[%d], Linds[%d], Lptrs[%d]" % (k, k, k))
# Add object terms to end of ode args string
for k in range(len(Lobj)):
    string_list.append("Lobj[%d]" % k)
for name, value in args.items():
    if isinstance(value, np.ndarray):
        string_list.append(name)
    else:
        string_list.append(str(value))
parameter_string = ",".join(string_list)

#
# generate and compile new cython code if necessary
#
if not opt.rhs_reuse or config.tdfunc is None:
    if opt.rhs_filename is None:
        config.tdname = "rhs" + str(os.getpid()) + str(config.cgen_num)
    else:
        config.tdname = opt.rhs_filename
    cgen = Codegen(h_terms=n_L_terms, h_tdterms=Lcoeff, args=args,
                   config=config)
    cgen.generate(config.tdname + ".pyx")
```

```
            code = compile('from ' + config.tdname + ' import cy_td_ode_rhs',
                           '<string>', 'exec')
            exec(code, globals())
            config.tdfunc = cy_td_ode_rhs

        #
        # setup integrator
        #
        initial_vector = mat2vec(rho0.full()).ravel('F')
        if issuper(rho0):
            r = scipy.integrate.ode(_td_ode_rhs_super)
            code = compile('r.set_f_params([' + parameter_string + '])',
                           '<string>', 'exec')
        else:
            r = scipy.integrate.ode(config.tdfunc)
            code = compile('r.set_f_params(' + parameter_string + ')',
                           '<string>', 'exec')
        r.set_integrator('zvode', method=opt.method, order=opt.order,
                         atol=opt.atol, rtol=opt.rtol, nsteps=opt.nsteps,
                         first_step=opt.first_step, min_step=opt.min_step,
                         max_step=opt.max_step)
        r.set_initial_value(initial_vector, tlist[0])

        exec(code, locals(), args)

        #
        # call generic ODE code
        #
        return _generic_ode_solve(r, rho0, tlist, e_ops, opt, progress_bar)

    def _td_ode_rhs_super(t, y, arglist):
        N = int(np.sqrt(len(y)))
        out = np.zeros(N, dtype=complex)
        y2 = np.zeros(len(y), dtype=complex)
        for i in range(N):
            out = cy_td_ode_rhs(t, y[i*N:(i+1)*N], *arglist)
            y2[i*N:(i+1)*N] = out
        return y2
```

```
# -----------------------------------------------------------------------------
# Master equation solver
#
def _mesolve_const(H, rho0, tlist, c_op_list, e_ops, args, opt,
                   progress_bar):
    """
    Evolve the density matrix using an ODE solver, for constant
hamiltonian
    and collapse operators.
    """

    if debug:
        print(inspect.stack()[0][3])

    #
    # check initial state
    #
    if isket(rho0):
        # if initial state is a ket and no collapse operator where given,
        # fall back on the unitary schrodinger equation solver
        if len(c_op_list) == 0 and isoper(H):
            return _sesolve_const(H, rho0, tlist, e_ops, args, opt,
                                  progress_bar)

        # Got a wave function as initial state: convert to density matrix.
        rho0 = ket2dm(rho0)

    #
    # construct liouvillian
    #
    if opt.tidy:
        H = H.tidyup(opt.atol)

    L = liouvillian(H, c_op_list)

    #
    # setup integrator
    #
    initial_vector = mat2vec(rho0.full()).ravel('F')
```

```
    if issuper(rho0):
        r = scipy.integrate.ode(_ode_super_func)
        r.set_f_params(L.data)
    else:
        r = scipy.integrate.ode(cy_ode_rhs)
        r.set_f_params(L.data.data, L.data.indices, L.data.indptr)
        # r = scipy.integrate.ode(_ode_rho_test)
        # r.set_f_params(L.data)
    r.set_integrator('zvode', method=opt.method, order=opt.order,
                     atol=opt.atol, rtol=opt.rtol, nsteps=opt.nsteps,
                     first_step=opt.first_step, min_step=opt.min_step,
                     max_step=opt.max_step)
    r.set_initial_value(initial_vector, tlist[0])

    #
    # call generic ODE code
    #
    return _generic_ode_solve(r, rho0, tlist, e_ops, opt, progress_bar)


#
# evaluate drho(t)/dt according to the master eqaution
# [no longer used, replaced by cython function]
#
def _ode_rho_func(t, rho, L):
    return L * rho

def _ode_rho_test(t, rho, data):
    # for performance comparison of cython code
    return data*(np.transpose(rho))
#
# Evaluate d E(t)/dt for E a super-operator
#

def _ode_super_func(t, y, data):
    ym = vec2mat(y)
    return (data*ym).ravel('F')

# ----------------------------------------------------------------------------
```

```
# Master equation solver for python-function time-dependence.
#
def _mesolve_func_td(L_func, rho0, tlist, c_op_list, e_ops, args, opt,
                     progress_bar):
    """
    Evolve the density matrix using an ODE solver with time dependent
    Hamiltonian.
    """

    if debug:
        print(inspect.stack()[0][3])

    #
    # check initial state
    #
    if isket(rho0):
        rho0 = ket2dm(rho0)

    #
    # construct liouvillian
    #
    new_args = None

    if len(c_op_list) > 0:
        L_data = liouvillian(None, c_op_list).data
    else:
        n, m = rho0.shape
        if issuper(rho0):
            L_data = sp.csr_matrix((n, m), dtype=complex)
        else:
            L_data = sp.csr_matrix((n ** 2, m ** 2), dtype=complex)

    if type(args) is dict:
        new_args = {}
        for key in args:
            if isinstance(args[key], Qobj):
                if isoper(args[key]):
                    new_args[key] = (
                        -1j * (spre(args[key]) - spost(args[key])))
```

```
            else:
                new_args[key] = args[key]
        else:
            new_args[key] = args[key]

elif type(args) is list or type(args) is tuple:
    new_args = []
    for arg in args:
        if isinstance(arg, Qobj):
            if isoper(arg):
                new_args.append((-1j * (spre(arg) - spost(arg))).data)
            else:
                new_args.append(arg.data)
        else:
            new_args.append(arg)

    if type(args) is tuple:
        new_args = tuple(new_args)
else:
    if isinstance(args, Qobj):
        if isoper(args):
            new_args = (-1j * (spre(args) - spost(args)))
        else:
            new_args = args
    else:
        new_args = args

#
# setup integrator
#
initial_vector = mat2vec(rho0.full()).ravel('F')
if issuper(rho0):
    if not opt.rhs_with_state:
        r = scipy.integrate.ode(_ode_super_func_td)
    else:
        r = scipy.integrate.ode(_ode_super_func_td_with_state)
else:
    if not opt.rhs_with_state:
        r = scipy.integrate.ode(cy_ode_rho_func_td)
```

```
        else:
            r = scipy.integrate.ode(_ode_rho_func_td_with_state)
        r.set_integrator('zvode', method=opt.method, order=opt.order,
                         atol=opt.atol, rtol=opt.rtol, nsteps=opt.nsteps,
                         first_step=opt.first_step, min_step=opt.min_step,
                         max_step=opt.max_step)
        r.set_initial_value(initial_vector, tlist[0])
        r.set_f_params(L_data, L_func, new_args)

        #
        # call generic ODE code
        #
        return _generic_ode_solve(r, rho0, tlist, e_ops, opt, progress_bar)


    #
    # evaluate drho(t)/dt according to the master equation
    #
    def _ode_rho_func_td(t, rho, L0, L_func, args):
        L = L0 + L_func(t, args)
        return L * rho


    #
    # evaluate drho(t)/dt according to the master equation
    #
    def _ode_rho_func_td_with_state(t, rho, L0, L_func, args):
        L = L0 + L_func(t, rho, args)
        return L * rho

    #
    # evaluate dE(t)/dt according to the master equation, where E is a
    # superoperator
    #
    def _ode_super_func_td(t, y, L0, L_func, args):
        L = L0 + L_func(t, args).data
        return _ode_super_func(t, y, L)

    #
```

```
# evaluate dE(t)/dt according to the master equation, where E is a
# superoperator
#
def _ode_super_func_td_with_state(t, y, L0, L_func, args):
    L = L0 + L_func(t, y, args).data
    return _ode_super_func(t, y, L)
```

3) Kode Simulator

```
from qutip import *
import numpy as np
import matplotlib.pyplot as plt

np.array(dir(np.math)[6:]) #operasi matematika yang dipakai

ustate = basis(3, 0) #Keadaan steady/tunak
excited = basis(3, 1) #Keadaan tereksitasi
ground = basis(3, 2) #keadaan dasar/ground

N = 2 # Menentukan besar matriks Hamiltonian

sigma_ge = tensor(qeye(N), ground * excited.dag())  # |g><e|
sigma_ue = tensor(qeye(N), ustate * excited.dag())  # |u><e|

a = tensor(destroy(N), qeye(3))
options = Options() #memanggil opsi untuk mesolve()

ada = tensor(num(N), qeye(3))
c_ops = []  # Build collapse operators
E_0 = 50
f = 1000
f2 = 1000
kappa = 0.5 # Cavity decay rate
c_ops.append(np.sqrt(kappa) * a)
gamma = 9  # Decay Rate
c_ops.append(np.sqrt(gamma) * sigma_ue)
c_ops.append(np.sqrt(gamma) * sigma_ge)
times = np.linspace(0, 150, 100) # Waktu total
psi0 = tensor(basis(N, 0), ustate) # State awal sistem
state_GG = tensor(basis(N, 1), ground) # State yang diproyeksikan,
yakni ground state (GG) dan steady state (UU)
sigma_GG = state_GG * state_GG.dag()
state_UU = tensor(basis(N, 0), ustate)
sigma_UU = state_UU * state_UU.dag()
g = 5  # konstanta coupling
```

```
H0 = -g * (sigma_ge.dag() * a + a.dag() * sigma_ge)  # Hamiltonian tanpa waktu
H1 = f - (sigma_ge.dag() + sigma_ge)  # Hamiltonian dengan waktu #1
H2 = f2 - (sigma_ue.dag() + sigma_ue)  # Hamiltonian dengan waktu #2
times2 = np.linspace(75, 150, 100) #Waktu yang digunakan laser untuk perpindahan ground -> excited
times3 = np.linspace(75,100, 100) #Waktu yang digunakan laser untuk perpindahan excited -> steady state (u)
options.nsteps = 1000000 #mendefinisikan step integrasi mesolve sebanyak satu juta langkah

def H0_coeff(radius):
    return 0.14 + (1/radius**2) * (1/0.13 + 1/0.45)

def H1_coeff(t2, t3):
    sigma = 8
    ta = 75
    I_a = E_0**2 / (np.sqrt(np.pi)*sigma) * np.exp(-(t3 - ta)**2/sigma**2) * np.exp(-(t3-t2)**2 / (4*sigma**2))
    E_tot = I_a
    return E_tot #Fungsi matematis laser pump

def H2_coeff(t, t3):
    sigma2 = 8
    tb = 50
    I_b = E_0**2 / (np.sqrt(np.pi)*sigma2) * np.exp(-(t - tb)**2/sigma2**2) * np.exp(-(t-t3)**2 / (4*sigma2**2))
    E_tot = I_b
    return E_tot #Fungsi matematis laser probe

H = [H0,[H1, H1_coeff(times, times3)], [H2, H2_coeff(times, times3)]]

result = mesolve(H, psi0, times, c_ops, [ada, sigma_UU, sigma_GG], options = options) #solver mesolve

fig, ax = plt.subplots()
ax.plot(times, result.expect[0]);
ax.set_xlabel('Delay Time (ps)');
ax.set_ylabel('Absorption rate');
```

4) Kode *recurrent neural network*

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from matplotlib import rc
from pandas.plotting import register_matplotlib_converters
from sklearn.preprocessing import RobustScaler
%matplotlib inline
%config InlineBackend.figure_format='retina'

register_matplotlib_converters()
sns.set(style='whitegrid', palette='muted', font_scale=1.5)

rcParams['figure.figsize'] = 22, 10

RANDOM_SEED = 42

np.random.seed(RANDOM_SEED)
tf.random.set_seed(RANDOM_SEED)

column_names = ['Target', 'Y', 'Number']

#df = pd.read_csv('C:\\Users\\Admin\\Documents\\xshkx-m78es.csv',
header=None, skiprows=[0], names=column_names)
df = pd.read_csv('C:\\Users\\Admin\\Documents\\9e5my-obiow.csv',
header=None, skiprows=[0], names=column_names)
#df_train = df
df_train = df[df['Number'] <= 23405]
df_test = df[df['Number'] > 23405]

#df_test = pd.read_csv('C:\\Users\\Admin\\Documents\\svxcj-jssz9.csv',
header=None, skiprows=[0], names=column_names)
```

```
#df_test.head

scale_columns = ['Y']

scaler = RobustScaler()

scaler = scaler.fit(df_train[scale_columns])

df_train.loc[:, scale_columns] = scaler.transform(df_train[scale_columns].to_numpy())
df_test.loc[:, scale_columns] = scaler.transform(df_test[scale_columns].to_numpy())

from scipy import stats

def create_dataset(X, y, time_steps=1, step=1):
    Xs, ys = [], []
    for i in range(0, len(X) - time_steps, step):
        v = X.iloc[i:(i + time_steps)].values
        labels = y.iloc[i: i + time_steps]
        Xs.append(v)
        ys.append(stats.mode(labels)[0][0])
    return np.array(Xs), np.array(ys).reshape(-1, 1)

TIME_STEPS = 200
STEP = 40

X_train, y_train = create_dataset(
    df_train[['Y']],
    df_train.Target,
    TIME_STEPS,
    STEP
)

X_test, y_test = create_dataset(
    df_test[['Y']],
    df_test.Target,
    TIME_STEPS,
    STEP
```

```
)

print(X_train.shape, y_train.shape)

from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(handle_unknown='ignore', sparse=False)

enc = enc.fit(y_train)

y_train = enc.transform(y_train)
y_test = enc.transform(y_test)

model = keras.Sequential()
model.add(
    keras.layers.Bidirectional(
      keras.layers.LSTM(
          units=128,
          input_shape=[X_train.shape[1], X_train.shape[2]]
      )
    )
)
model.add(keras.layers.Dropout(rate=0.5))
model.add(keras.layers.Dense(units=128, activation='relu'))
model.add(keras.layers.Dense(y_train.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy',          optimizer='adam',
metrics=['acc'])

history = model.fit(
    X_train, y_train,
    epochs=12000,
    batch_size=64,
    validation_split=0.1,
    shuffle=True
)

plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend();
```

```
y_pred = model.predict(X_test)

from sklearn.metrics import confusion_matrix

def plot_cm(y_true, y_pred, class_names):
  cm = confusion_matrix(y_true, y_pred)
  fig, ax = plt.subplots(figsize=(12, 10))
  ax = sns.heatmap(
      cm,
      annot=True,
      fmt="d",
      cmap=sns.diverging_palette(220, 20, n=7),
      ax=ax
  )

  plt.ylabel('Actual')
  plt.xlabel('Predicted')
  ax.set_xticklabels(class_names)
  ax.set_yticklabels(class_names)
  b, t = plt.ylim() # discover the values for bottom and top
  b += 0.5 # Add 0.5 to the bottom
  t -= 0.5 # Subtract 0.5 from the top
  plt.ylim(b, t) # update the ylim(bottom, top) values
  plt.show() # ta-da!

plot_cm(
  enc.inverse_transform(y_test),
  enc.inverse_transform(y_pred),
  enc.categories_[0])
```

5) Datasheet Spektrofotometer MayaPRO2000

830 Douglas Ave.
Dunedin, FL 34698
(727)733-2447
Fax(727)733-3962
www.OceanOptics.com

Ocean Optics

A HALMA GROUP COMPANY

Maya2000Pro Data Sheet for Versions 3.00.1 and Above

Description

The Ocean Optics Maya2000Pro and Maya2000Pro-NIR include the linear CCD-array optical bench, plus all the circuits necessary for spectrometer operation. The result is a compact, flexible system, with no moving parts, that's easily integrated as an OEM component.

Note

This data sheet is for Maya2000Pro FPGA and FX2 firmware version 3.00.1 and above. For Maya2000Pro Spectrometers with firmware below this version, and for the Maya2000 Spectrometer, please see the Maya2000 and Maya2000Pro Data Sheet.

The Maya2000Pro and Maya2000Pro-NIR Spectrometers are a unique combination of technologies providing users with high sensitivity for low light-level, UV-sensitive and other scientific applications. The electronics have been designed for considerable flexibility in connecting to various modules as well as external interfaces. The Maya2000Pro series spectrometers interface to PCs, PLCs and other embedded controllers through USB 2.0 or RS-232 communications.

The detector used in the Maya2000Pro series spectrometers are a scientific-grade, back-thinned, CCD array from Hamamatsu (product number S10420 for Maya2000Pro and S11510 for Maya2000Pro-NIR). For complete details on these detectors, visit www.Hamamatsu.com.

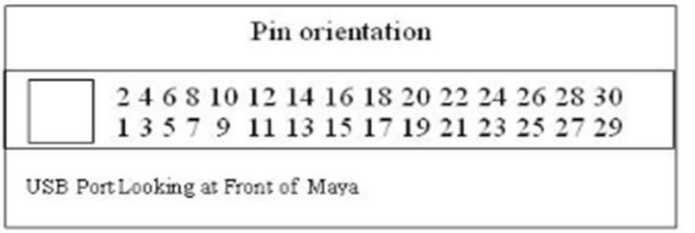
The Maya operates from power provided through the USB, or from a separate + 5VDC power supply. The Maya is a microcontroller-controlled spectrometer, thus all operating parameters are implemented through software interfacing to the unit.

Features

- Back-thinned Hamamatsu detectors
  - S10420 – high UV Sensitivity in Maya2000Pro
  - S11510 – high NIR Sensitivity in Maya2000Pro-NIR
- Spectrometer Design:
  - Symmetrical Crossed Czerny Turner
  - 101.6 mm focal length
  - 15 gratings including the HC-1 composite grating
  - 6 slit widths
- Electrical Performance:
  - 16 bit, 500kHz A/D Converter
  - Integration time: 7.2ms – 5s
- Embedded microcontroller allows programmatic control of all operating parameters and standalone operation:
  - USB 2.0 480Mbps (high-speed) and 12Mbps (full speed)
  - RS232 115Kbaud
  - Communication Standards for digital accessories (I2C)
- Onboard Pulse Generator:
  - 2 programmable strobe signals for triggering other devices
  - Software control of nearly all pulse parameters
- Onboard GPIO:
  - 10 user-programmable digital I/O
- EEPROM storage for:
  - Wavelength Calibration Coefficients
  - Linearity Correction Coefficients
  - Absolute Irradiance Calibration (optional)
- Plug-n-Play Interface for PC applications
- 30-pin connector for interfacing to external products

Electrical Pinout

Listed below is the pin description for the Maya Accessory Connector (J3) located on the front vertical wall of the unit. The connector is a Pak50TM model from 3M Corp. Headed Connector Part# P50-030P1-RR1-TG. Mates with part# P50-030S-TGF (requires two: 1.27mm (50 mil) flat ribbon cable: Recommended 3M 3365 Series)



Pin #	Function	Input/Output	Description
1	RS232 Rx	Input	RS232 receive signal – Communicates with a PC over DB9 F
2	RS232 Tx	Output	RS232 transmit signal – Communicates with a PC over DB9
3	GPIO (2)	Input/Output	Reserved
4	N/A	N/A	Reserved
5	Ground	Input/Output	Ground
6	I2C SCL	Input/Output	I2C clock signal for communication to other I2C peripherals
7	GPIO (0)	Input/Output	Base clock
8	I2C SDA	Input/Output	I2C data signal for communication to other I2C peripherals
9	GPIO (1)	Input/Output	Master clock
10	Ext. Trigger In	Input	TTL input trigger signal
11	GPIO (3)	Input/Output	Integration clock
12	VCC or 5VIN	Input or Output	Input power pin for Maya – When operating via USB, this pin power other peripherals – Ensure that peripherals comply wit USB specifications
13	N/A	N/A	Reserved
14	VCC or 5VIN	Input or Output	Input power pin for Maya – When operating via USB, this pin power other peripherals – Ensure that peripherals comply wit USB specifications
15	SPI Data In	Input	SPI Master In Slave Out (MISO) signal for communication to SPI peripherals
16	GPIO (4)	Input/Output	Reserved

5) Datasheet Newport DL325

DL Series

High-Performance Delay Line Stages

mks | Newport

The DL linear stage series is a high performance but very affordable, linear motor driven stage with an integrated motion controller. Optimized for small loads, repeatable positioning and fast traverse speeds, it is an ideal solution for spectroscopy applications that require delay lines. With travels of 125 mm, 225 mm and 325 mm, this offering covers almost all possible delay needs from femtosecond to nanosecond delays. Spectroscopy applications range from pump-probe, interferometry, 2DIR, etc. To facilitate setups, beam kits consisting of retroreflectors, mirrors, mounts and other optomechanical parts, are available to suit various wavelengths and delay line configurations .

Design Details

- Base Material: Extruded Aluminum
- Bearings: Recirculating bearings
- Drive System: 3 phase synchronous ironless linear motor (without Hall effect sensors)
- Motor Initialization: Done by the controller.
- Motor Commutation: Done by the controller on encoder feedback
- Feedback: Linear glass scale, 80 µm signal period, 1 VPP
- Limit: Optical
- Home Switch: Optical, on encoder's fiducial track, located at the minus end of travel
- Controller: DL Controller
- Cable: 3 m long pigtail cables included
- MTBF: 20,000 hours

Key Features

- Excellent delay sensitivity and bi-directional repeatability
- Low angular deviation where it counts (pitch)
- Compatibility with optical tables & mounts
- Small footprint
- No moving cable
- Easy to use (Delay line GUI, LabVIEW drivers)

Specifications

		DL125	DL225	DL325
Travel Range (Single Pass)	(mm)	125	225	325
	(ns)	0.8	1.5	2.2
Minimum Incremental Motion (Single Pass)	(mm)	75	75	75
	(fs)	0.5	0.5	0.5
Bi-direct. Repeatability, Guar.[1] (µm)		±0.15	±0.15	±0.15
Accuracy, Guaranteed[1] (µm)		±1.5	±2	±2.5
Encoder Resolution (nm)		50	50	50
Origin Repeatability (µm)		0.4	0.4	0.4
Maxi. Speed[2] (mm/s)		500	500	500
Max. Acceleration, No Load (mm/s²)		7500	7500	7500
Pitch, Typical (Guaranteed)[2,3] (µrad)		±60 (±100)	±60 (±100)	±90 (±150)
Yaw, Typical (Guaranteed)[2,3] (µrad)		±30 (±60)	±40 (±90)	±50 (±120)

[1] *For the definition of Typical and Guaranteed specifications see "Motion Basics Terminology & Standards" Tutorial at www.newport.com*

[2] *With DL controller.*

[3] *To obtain arcsec units, divide µrad value by 4.8.*

		Single Pass	Dual Pass	Quad Pass
Delay	DL125 (ns)	0.8	1.7	3.3
	DL225 (ns)	1.5	3.0	6.0
	DL325 (ns)	2.2	4.3	8.7
MIM	(fs)	0.5	1.0	2.0

2.0 Description

The DL linear stages Series is a high performance but very affordable, linear motor driven with an integrated motion controller. Optimized for small loads, repeatable positioning and fast traverse speeds, it's an ideal solution for spectroscopy applications ranging from pump-probe, interferometry, 2DIR, etc.

With travel ranges of 125 mm, 225 mm and 325 mm, this offering covers almost all possible delay needs from femtosecond to nanosecond delays

The DL stage utilizes an FEM-optimized extruded aluminum body that is extremely stiff, while minimizing the bending effect caused by the differe thermal expansion coefficients of the aluminum body and the steel rails. The rails' position relative to the profile's neutral fiber minimizes the eff due to bi-metal thermal expansion. The body's rigidity minimizes the deflection under load.

The stage has been designed to be mounted on standard breadboards (with a 0.2 mm flatness). This minimizes the effect of the table to pitch, yaw and roll performance of the stage.

Recirculating ball bearing slides provide excellent payload capabilities a long life. The movement is smooth with low noise.

Unlike screw driven stages, the DL employs a centered, high efficiency 3- phase, synchronous ironless, linear motor as the driving element. This drive system is absolutely non-contact, and has the advantage of higher speed, high acceleration and high system responsiveness without wear associated with motor brushes or drive screws. And because of the fully integrated linear motor, the DL is shorter than a comparable screw drive stage.

Precision position feedback is provided by a very repeatable linear scale mounted in the stage. The encoder signals are interpolated by the dedicated Newport motion controller with nanometer resolution for outstanding position sensitivity, repeatability, and stability. A home position is incorporated on the same scale, avoiding the use of additiona electronics or mechanics for improved reliability and accuracy.

Thus, the DL is the optimum solution for space constrained applications that require high-throughput, high reliability, and ultra-quiet operation.

To facilitate setups, beam kits consisting of retroreflectors, mirrors, mounts and other optomechanical parts, are available to suit various wavelengths and delay line configurations.

2.1 Design Details

Base Material	Extruded Aluminum
Bearings	Recirculating bearings
Drive System	3 phase synchronous ironless linear motor (without Hall effect sensors)
Motor Initialization	Has to be done by the controller
Motor Commutation	Done by the controller on encoder feedback
Feedback	Linear glass scale, 80 µm signal period, 1 V_{pp}
Limit	Optical
Home Switch	Optical, on encoder's fiducial track, located at the minus end of travel
Controller Compatibility	DL Controller
Cable	3 m long pigtail cables included
MTBF	20,000 hours

3.5 Dynamic Characteristics

Dynamic response of the stage depends on two parameters:

- The force delivered by the motor
- The total moving mass including carriage mass and payload

Newport's DL Controller includes four parameters allowing dynamic control optimization of the stage:

- Scaling Force
- Force Limit
- Carriage Mass
- Payload Mass

Although the three parameters are set in the factory perstage, the fourth (Payload Mass) must be adjusted by the user. Refer to the "FM" command of the Newport DL Controller user's manual.

3.5.1 Description of Parameters

The DL Controller/driver "ScalingForce" parameter is used by the controller to scale the output current. It indicates the theoretical maximum force (friction not taken in account) of the DL carriage with the controller full scale current (See Controller Manual).

The DL Controller/driver "ForceLimit" parameter indicates the maximum force of the DL stage carriage limited by current saturation to avoid stage damage.

Then, stage carriage acceleration can be calculated using the following formula:

$$\textbf{MaxAccelaration} = \frac{\textbf{ForceLimit - Friction Constant}}{\textbf{Carriage Mass + Payload mass}}$$

See chapter 3.4.3 acceleration/Load graph.

- Due to its linear motor and rails, DL stage Friction Constant is only 4 N.
- **Carriage Mass** is **0.85 kg**.

3.5.2 Stages Adjustment Proposal in Function of the Load

In order to improve and have better driving performances, it is advisable to change the control parameters according to the load. All the stages are delivered with a crossover frequency of 70 Hz. You can change the parameters KP, KI, KD, FD, FL and FMP in the controller terminal. Please see the following table:

Customer payload from High (7 kg) to low (0 kg) FMP command	High	Medium	Low
Response Time	Slow	Medium	Fast
Crossover Frequency (Hz)	55	70	80
KP	19,000	31,000	48,000
KI	2,000,000	3,500,000	5,000,000
KD	350	420	500
Derivative cut off frequency (FD)	100	140	150
PID 2nd Order low pass filter (FI)	4,000	4,000	4,000

3.6 Stage Weights

The stage weight below does not include the cable.

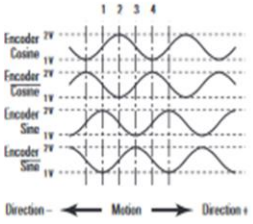
	Weight [lb (kg)]
DL175	7.1 (3.2)
DL275	8.8 (4.0)
DL375	10.4 (4.7)

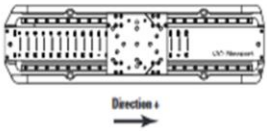
4.0 Motor & Feedback

4.1 Brushless Motor Characteristics

	Motor Constant (N²/W)	Magnet Pitch (mm)	Nominal Voltage (V)	Max. RMS Current (A)	Max. Peak Current (A)	Resistance per Phase (Ω)	Inductance per Phase (mH)
DL Stage	4.2	30	42	1.3	4.2	4	0.72

4.2 Feedback Signal Position

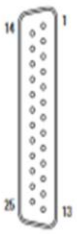




4.3 Pinouts

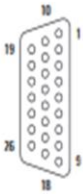
The pinout diagrams for DL stage connectors are shown below.

Motor Connector (SUB-D25M)



1	U Motor	14	Ground
2	U Motor	15	N.C.
3	N.C.	16	Ground
4	N.C.	17	Reserved (SMDAT)
5	V Motor	18	Reserved (SMCLK)
6	V Motor	19	N.C.
7	W Motor	20	N.C.
8	W Motor	21	+5 V
9	Reserved (Thermistor)	22	Ground
10	N.C.	23	N.C.
11	N.C.	24	N.C.
12	N.C.	25	N.C.
13	N.C.		

Encoder Connector (HD26M)



1	+5 V	14	+ End-of-Run
2	N.C.	15	/Stne
3	N.C.	16	N.C.
4	Costne	17	Index Pulse /I
5	- End-of-Run	18	N.C.
6	Stne	19	+ 5 V
7	Ground	20	N.C.
8	Index Pulse I	21	N.C.
9	N.C.	22	N.C.
10	N.C.	23	N.C.
11	N.C.	24	N.C.
12	N.C.	25	Ground
13	/Costne	26	Ground

Halaman ini sengaja dikosongkan.

BIODATA

Reynaldi Gilang Mulyawan lahir di Surabaya pada tanggal 20 April 1996 dan memiliki *passion* yang kuat di bidang riset, khususnya riset pada ilmu fisika. Sebelum saya menempuh pendidikan di Departemen Teknik Elektro, Fakultas Teknologi Elektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember angkatan 2016, saya menempuh pendidikan di SMA MIMI dari tahun 2011 hingga 2014. Saya sempat mengenyam pendidikan sebagai mahasiswa pertukaran pelajar di Universiti Teknologi Malaysia pada semester genap tahun ajaran 2017/2018, serta telah menjadi *co-author* pada dua jurnal ilmiah, yang berjudul "*Color Conversion of Caramelized Sugar Carbon Dots Using Blue and Green Light Emitting Diodes"* dan "*Simple Application of Time Correlated Single Photon Counter of Picosecond Pulsed Laser to MeasureRefractive Index of Saline Solution"*, masing-masing diterbitkan sebagai *proceeding* untuk *International Symposium of Frontiers in Applied Physics (ISFAP) 2019* dan Jurnal Penelitian Fisika dan Aplikasinya, Universitas Negeri Surabaya (UNESA)