Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 6, No. 11, November 2022, hlm. 5413-5421 http://j-ptiik.ub.ac.id

Navigasi Halang Rintang Quadcopter Tanpa Awak menggunakan RPLIDAR dan Algoritma Bug2 berbasis Raspberry Pi

Mochammad Zava Abbiyansyah[1], Eko Setiawan[2], Agung Setia Budi[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]abbyzava0062@gmail.com, [2]ekosetiawan@ub.ac.id, [3]agungsetiabudi@ub.ac.id

Abstrak

Kompetisi kontes Robot Terbang Vertical takeoff landing yang merupahan salah satu cabang lomba di Kontes Robot terbang Indonesia (KRTI) tahun 2021 yang lalu, memberikan salah satu tantangan baru yaitu terdapat satu buah rintangan yang akan menghalangi pergerakan dari robot terbang yang di ibaratkan seperti Gedung yang penempatannya akan dilakukan secara acak dan perlombaan di adakan pada dalam ruangan yang menyebabkan nilai dari GPS dan Barometer tidak dapat digunakan. Sistem ini menggunakan flight controller Pixhawk, Raspberry Pi 4, RPLIDAR, OpticalFlow, Ultrasonic GY-US42 dan frameworks robot operaring system (ROS). Algoritma Bug2 dapat menjadi solusi karena memiliki proses yang ringan dan hanya membutuhkan parameter masukan dari jarak sensor Lidar dan posisi global dari sensor OpticalFLow. Penerapan algoritme Bug2 menggunakan nilai kecepatan pada 1 m/s dan jarak diteksi 1.5 m sesuai pada pengujian dengan melakukan simulasi pada Gazebo. Untuk deteksi halang rintang, algortima Bug2 mendapatkan hasil keberhasilan pada 90%, waktu tempuh sebesar 24.3 detik, dan penggunaan CPU sebesar 63%.

Kata kunci: *Quadcopter, ROS, Gazebo, Halang Rintang, Algoritma Bug2, KRTI.*

Abstract

The Vertical takeoff landing Flying Robot Contest competition, which is one of the branches of the Indonesian Flying Robot Contest (KRTI) in 2021, provides a new challenge, namely that there is an obstacle that will hinder the movement of the flying robot which is likened to a building whose placement is will be carried out randomly and the competition will be held indoors causing the GPS and Barometer values to be unusable. This system uses the Pixhawk flight controller, Raspberry Pi 4, RPLIDAR, OpticalFlow, Ultrasonic GY-US42 and robot operating system (ROS) frameworks. The Bug2 algorithm can be a solution because it has a lightweight process and only requires input parameters from the Lidar sensor distance and global position from the OpticalFlow sensor. The application of the Bug2 algorithm uses a speed value of 1 m/s and a detection distance of 1.5 m according to the test by simulating the Gazebo. For obstacle detection, the Bug2 algorithm gets success results at 90%, a travel time of 24.3 seconds, and CPU usage of 63%.

Keywords: *Quadcopter, ROS, Gazebo, Obstacle Avoidence, Bug2 Algorithm, KRTI.*

1. PENDAHULUAN

Kompetisi rancang bangun dari *drone* yang dilaksanakan oleh Pusat Prestasi Nasional (PUSPRESNAS) kementrian pendidikan, kebudayaan, Riset, dan Teknologi (Kemendikbud Ristek) yang dikenal dengan nama Kontes Robot Terbang Indonesia (KRTI), merupakan suatu lomba yang setiap tahun diselenggarakan sebagai realisasi pada perkembangan *drone* atau UAV yang ada di Indonesia. Tahun 2021 yang lalu sudah dilaksanakan kompetisi KRTI pada berbagai divisi. Salah satunya yaitu cabang VTOL (*Vertical TakeOff Landing*) yang menggunakan *drone* berjenis *quadcopter*. Pada cabang ini *quadcopter* harus dapat membawa barang dan mengirimkannya pada tempat yang sudah ditentukan dengan halangan berupa kotak-kotak tinggi yang diibaratkan sebagai gedung. Area kompetisi yang berbeda dengan tahun-tahun sebelumnya ini yang membuat kemampuan dari *quadcopter* perlu ditingkatkan untuk dapat mendeteksi halangan disekitarnya.

Drone harus dapat melewati halangan

tersebut dengan cepat dan tepat. Penggunaan sensor LiDAR pada beberapa sisi seperti pada *drone* konvensional memiliki kekurangan pada jangkuan sisi yang dapat dijangkau. Maka penggunaan sensor RPLIDAR A1 atau sensor LiDAR 360 merupakan salah satu pilihan terbaik untuk menjangkau semua sisi pada *drone* (Xuexi *et al.*, 2019).

Prinsip dasar dari algoritma *Bug2* adalah algoritma ini tidak menyadari hambatan di lingkungan mereka dan hanya menyadari posisi relatif dari target yang akan dituju (McGuire, de Croon and Tuyls, 2019), hal ini akan membuat *drone* selalu berfokus pada posisi akhir membuat waktu yang didapat pada perlombaan KRTI akan semakin cepat. Algoritma Bug secara ideal cocok untuk navigasi dalam ruangan pada sistem robot dengan sumber daya terbatas, karena potensi memori dan persyaratan pemrosesannya rendah (Divya *et al.*, 2021) sesuai dengan karakteristik pada area kompetisi KRTI. Pada penelitian yang dilakukan oleh (Xu, Yu and Bai, 2017) dengan menggunakan robot beroda mendapatkan waktu rata-rata untuk menyelesaikan satu halang rintang selama 1.32 detik dengan kecepatan rata-rata menunjukkan 1.3 m/s. Penelitian diatas menggunakan algotitma *bug 0*, *bug 1*, dan *bug 2* sebagai perbandingan. Untuk hasil tercepat didapatkan pada algoritma *bug 2* dengan waktu selisih 1,5 detik disetiap algoritma bug.

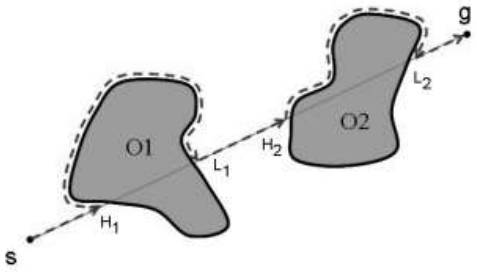
Pada penelitian yang akan dilakukan oleh Penulis, Penulis ingin mengembangkan penelitian sebelumnya dengan melakukan percobaan pada deteksi halang rintang menggunakan pesawat nir awak segi empat berbasis pada Pixhawk dengan menambahkan sensor RPLIDAR A1 pada bagian atas *drone*, serta memberikan mini computer berupa Raspberry Pi 4 untuk dimuat pada *drone*. Pengambilan *drone* berbasis Pixhawx karena *flight controller* bersifat *open source*, serta menggunakan sensor RPLIDAR A1 karena memiliki daya jangkauan 360° untuk dapat mengjangkau keseluruhan sisi *drone*. Sehingga membutuhkan sedikit ruang pada komputer terpasang, dan akan membuat sumber daya tersedia untuk pekerjaan lain. Algoritma *bug* digunakan karena memiliki kemampuan dalam komputasi yang rendah. Penggunaan algoritma *bug* yang dilakukan oleh (Xu, Yu and Bai, 2017), dengan berbasis robot beroda dengan menggunakan sensor Ultrasonic HC SR04 dengan kecepatan 1.32 detik.

2. Dasar Teori

2.1 Algoritma Bug2

Algoritma perencanaan gerak sederhana dikenal sebagai algoritma *bug*. Algoritma *Bug* memecahkan masalah navigasi dengan menyimpan sejumlah titik arah minimal dan tidak menghasilkan peta lingkungan yang komprehensif (Arif *et al.*, 2020). Algoritma *Bug2* adalah algoritma yang berkinerja baik dalam keadaan tertentu. *Bug2* akan mengingat jalur dari posisi awal ke tujuannya (Yang *et al.*, 2019).

Kontur rintangan dimulai pada titik H1 dalam algoritma Bug 2 dan berakhir setiap kali robot melewati garis ke tujuan. Ini menetapkan batas rintangan-mengikuti perilaku meninggalkan titik L1. Robot melakukan perjalanan langsung ke tujuan dari L1. Jika lebih banyak hambatan ditemukan, proses diulang. Jalur yang dibuat oleh Bug 2 dengan dua hambatan ditunjukkan pada Gambar 1. Diteksi Halangan Algoritma Bug2 Secara umum, algoritma Bug 2 berjalan lebih cepat daripada algoritma Bug 1 dan lebih efektif, terutama di area terbuka. Namun, ada beberapa keadaan di mana itu mungkin bukan pilihan terbaik. Secara khusus, konstruksi seperti labirin dapat menjebak robot. Kesederhanaan algoritma ini menyebabkan berbagai kelemahan (Moysis *et al.*, 2020).



Gambar 1. Diteksi Halangan Algoritma Bug2

Sumber : (Ribeiro, 2005)

Rintangan yang sudah diketahui robot atau yang ditemukan oleh sensor onboardnya dijauhkan karena kecenderungannya untuk menolak. Kekuatan potensi tolak-menolak ini meningkat dengan kedekatan robot dengan rintangan dan berkurang dengan jaraknya (Mohsen, Sharkas and Zaghlol, 2019). Mengingat bahwa masalahnya adalah linier, efek penolak total dari semua hambatan menentukan potensi penolakan (Pozna *et al.*, 2022).

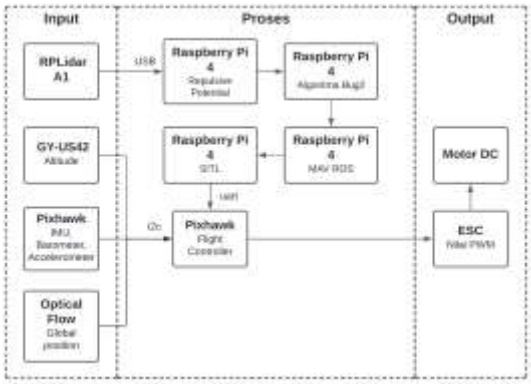
Halangan yang sangat jauh tidak mungkin menghalangi robot. Terlebih lagi, ketika robot mendekati rintangan, kekuatan potensi tolakan harus tumbuh. Potensi tolak-menolak potensial yang dihasilkan oleh rintangan I dipertimbangkan untuk menjelaskan hasil ini serta efek-efek ruang-terikat (Blok *et al.*, 2019).

$$u = \frac{1}{2} K_{obstacle_i} \left(\frac{1}{d_{obstacle_i}(q)} - \frac{1}{d_0}\right)^2 \quad (1)$$

Pada Persamaan (1) merupakan algoritma bug2 yang melakukan perhitungan dari nilai $K_{obstacle_i}$ yang merupakan jarak toleransi minimal dari objek halangan, nilai $d_{obstacle_i}$ yang merupakan hasil pembacaan jangkauan, dan nilai d_0 yang merupakan jangkauan toleransi maksimal dari objek halangan.

3. PERANCANGAN DAN IMPLEMENTASI

3.1 Gambaran Umum Sistem



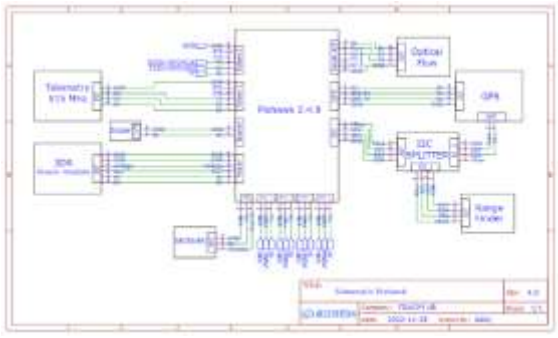
Gambar 2. Diagram Blok Sistem

Pada Gambar 2 merupakan blok diagram dari keseluruhan system yang dibagi menjadi tiga bagian, yaitu *input*, proses, dan *output*. Pada bagian *input* system terdiri atas 4 bagian, yaitu sensor RPLidar A1, GY-US42, Pixhawk, dan Optical Flow. Peran dari keempat sensor ini memiliki fungsinya masing-masing sepetri membaca ketinggian langsung *drone*, hingga mengetahui posisi Global dari *drone*. Pada bagian proses terdiri atas beberapa bagian utama yaitu pemrosesan *input* dari sensor RPLidar menggunakan metode *Repulsive Potential* dan algoritma *Bug2* sebagai algoritma halang rintang yang diproses pada Jetson Nano. Hasil dari proses akan dikirimkan melalui *MAV ROS* yang kemudian dikirimkan kepada Pixhawk melalui komunikasi Mavlink melalui koneksi UART. Output pada system akan mengeluarkan sinyal PWM untuk mengendalikan laju dari motor DC system

3.2 Perancangan Perangkat Keras

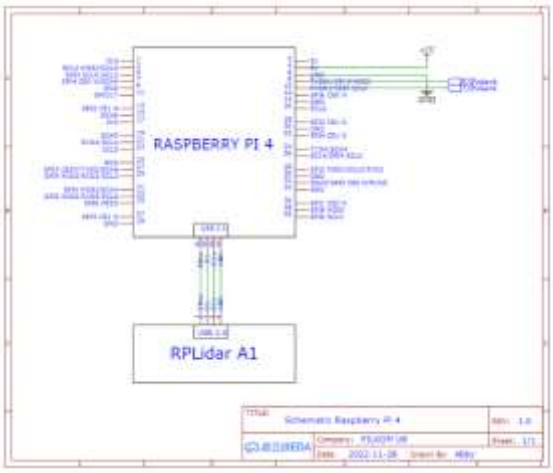
Gambar 3. Desain Tiga Dimensi Robot Terbang

Robot terbang dirancang dengan mengadopsi model *quadcopter*, yang memiliki empat baling-baling. Pada Gambar 3 merupakan desain tiga dimensi dari robot terbang *quadcopter* yang sudah dilakukan modifikasi pada penempatan sensor RPLIDAR pada bagian atas. Pada gambar diatas dapat dilihat pula bagian dari Raspberry Pi 4 sebagai tempat komputasi dan Pixhawk sebagai *flight controller*.



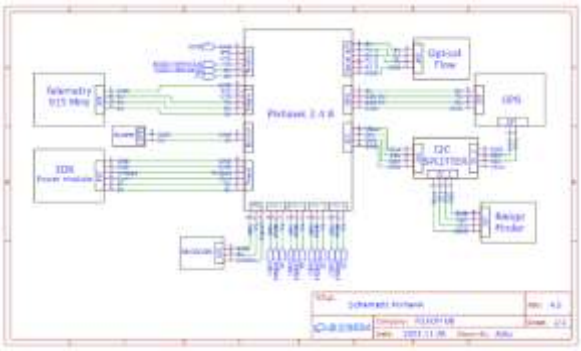
Gambar 4. Skematik System Pixhawk

Komponen pada diagram blok disusun dengan mempertimbangkan fungsi kegunaan dari masing-masing komponen. Seperti pada Gambar 4 yang menggambarkan skematik dari sistem Pixhawk 2.4.8. Semua komponan akan terhubung pada konektor yang berada pada pixhawk sesuai dengan fungsinya masing-masing. Untuk menghubungkan raspberry pi dengan Pixhawk melalui konektor Telem2 dengan menggunakan port GND, RX, TX, dan 5V sebagai daya dari raspberry pi.



Gambar 5. Skematik System Raspberry Pi 4

Gambar 5 merupakan skematik dari system raspberry Pi 4 yang digunakan sebagai system komputasi dari penelitian ini. Modul RPLidar A-1 terhubung langsung dengan raspberry pi 4 dengan menggunakan kabel Micro USB. Sistem komunikasi Pixhawk dengan raspberry pi akan terhubung pada *pinout* yang terdapat pada Raspberry Pi.



Gambar 6. Skematik System Pixhawk

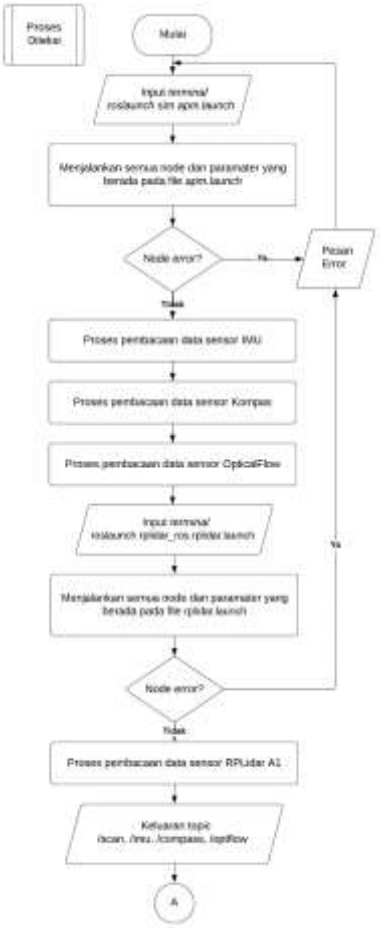
Berdasarkan skemati sistem pada Pixhawk yang sudah dirancang, maka dapat dilakukan konfigurasi pin-pin yang terdapat pada Pixhawk untuk dapat melakukan integrasi dengan sensor-sensor serta perangkat komunikasi Pixhawk. Konfigurasi pin yang dirancang akan menjelaskan dengan detail terkait pin-pin yang digunakan perangkat, berdasarkan skemati sistem yang terhubung pada Pixhawk sesuai dengan Gambar 5.3.

3.3 Perancangan Perangkat Lunak

Gambar 7. Diagram Alir Sistem

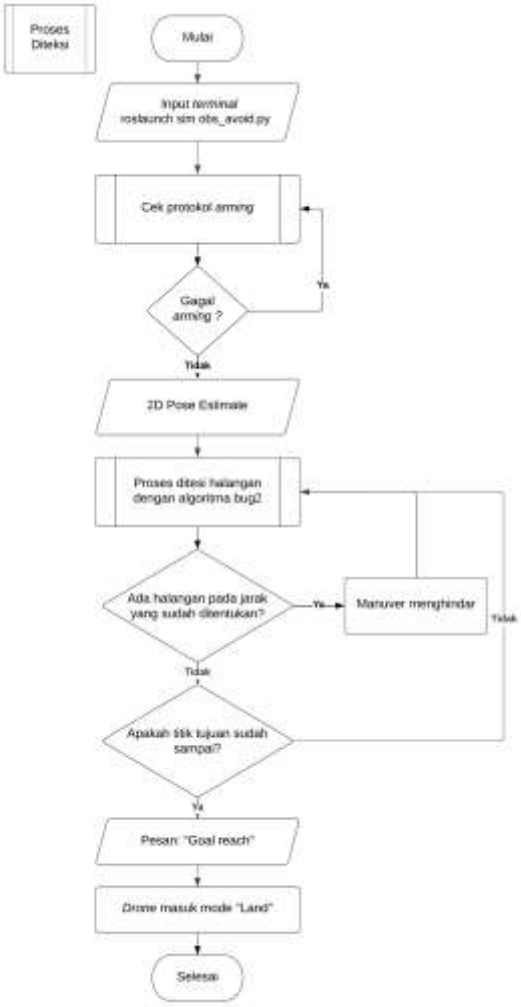
Gambar 7 menjelaskan terkait perencanaan diagram alir dari keseluruhan program utama. Proses perancangan perangkat lunak dilakukan menggunakan komputer sebagai alat simulator utama, sedangkan pada raspberry pi digunakan sebagai tempat *deploy* keseluruhan program. Pada proses persiapan Raspberry Pi4 dilakukan menggunakan protocol MavLink untuk menghubungkan dengan Pixhawk. Pada proses kalibrasi Pixhawk ditujukan untuk memastikan seluruh sensor-sensor dapat berjalan dengan baik sesuai dengan protocol yang telah diterapkan oleh Pixhawk. Sensor kompas, IMU, serta OpticalFlow akan dilakukan kalibrasi karena, pada setiap peletakkannya membutuhkan nilai yang berbeda-beda pula tergantung konfisi ruangan dan sekitar. Pada tahap proses diteksi dilakukan untuk merancang system agar dapat berjalan baik dengan algoritma Bug2. Terahir pada proses simulasi Gazebo keseluruhan program yang telah dirancang dapat di coba dengan menggunakan SITL sebagai simulator dari *drone* dan Gazebo sebagai simulator *visual* pada *drone*.

Proses rerancangan diteksi memiliki beberapa bagian yang memiliki fungsinya masing-masing, pada Gambar 8 merupakan diagram alir dari proses perancangan diteksi.



Gambar 8. Diagram Alir Perancangan Proses Diteksi Awal

Proses ini diawali dengan menjalankan perintah *$ roslaunch sim apm.launch* pada terminal Raspberry, proses ini berfungsi sebagai perintah untuk menjalankan Mavros. Setelah proses ini di jalankan maka, pada *$ rostopic* list akan muncul topic yang di keluarkan oleh Mavros yang berasal dari Pixhawk. Begitu pun dengan sensor kompass dan OpticalFlow yang memiliki nodenya masing-masing. Selanjutnya akan di jalankan kode perintah *$ roslaunch rplidar_ros rplidar.launch* pada terminal baru untuk menjalankan ros pada sensor RPLidar A1. Perintah tersebut akan mengeluarkan *topic /scan* sebagai keluaran dari sensor RPLidar A1 yang merupakan data raw yang belum di proses. *Topic /scan* akan digunakan untuk menjadi masukan utama dari system diteksi halangan, yang akan di kombinasikan dengan sensor-sensor yang terdapat pada Pixhawk. Apabila semua *topic* di atas sudah dapat terbaca dengan baik, yang ditandai dengan pesan dari Mavros yang telah dapat membaca versi dan tipe dari drone yang digunakan, yaitu versi *copter* V4.3.



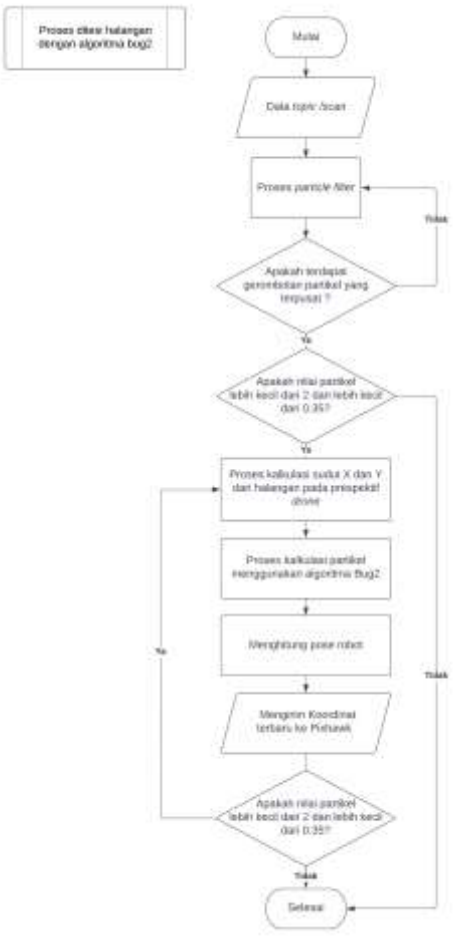
Gambar 9. Diagram Alir Perancangan Proses Diteksi Lanjutan

Gambar 9 merupakan diagram alir dari perancangan proses diteksi lanjutan. Proses ini dimulai dengan cek protokol *arming* merupakan proses yang wajib di lakukan sebelum *drone* dapat lepas landas. Semua parameter dari pembacaan sensor, sistem *flight controller*, sistem koneksi telemetri, koneksi dengan *remote control*, dan beebagai macam parameter keamanan lainnya akan di lakukan pengecekan sesuai dengan *range* toleransi yang sudah di tentukan. Apabila terdapat salah satu *error* pada parameter yang digunakan, *drone* tidak akan dapat masuk kedalam mode *arming*. Maka, apabila terjadi *error* Pixhawk harus dilakukan *tracing problem* terkait parameter yang terdapat error. Apabila berhasil maka *drone* sudah dapat *arming* dan dapat menjalankan program utama.

Selanjutnya *drone* akan memunculkan koordinat *2d pose estimate* dari hasil pembacaan OpticalFlow yang akan secara otomatis menjadi titik nol dari pergerakan *drone* selanjutnya. Pada

proses ini *drone* sudah memasuki mode "*guided*" untuk siap menerima perintah masukan dari Raspberry Pi. Pada proses diteksi halangan menggunakan masukan sensor RPLidar A1 dengan memanggil topic */scan*, *topic* ini akan menampilkan jarak halangan objek secara 2 dimensi. Diagram alir dari proses diteksi halangan menggunakan algoritma Bug2 dapat dilihat pada Gambar 9.

Gambar 10. merupakan tahap diteksi halangan menggunakan algoritma Bug2 yang di mulai dengan membaca nilai data dari *topic /scan* pada sensor RPLidar. Nilai tersebut merupakan partikel-partikel yang secara random tersebar dengan nilai jarak pada masing-masing partikel. *Particle filter* menggunakan data jarak dari sensor RPLidar untuk dapat melakukan lokalisasi sekitar. Apabila terdapat particle yang terkumpul secara terpusat maka hal ini dapat dikatakan sebagai halangan.



Gambar 10. Diagram Alir Proses Diteksi Halangan Dengan Algoritma Bug2

Untuk mengetahui sudut x dan y dari prespektif *drone* di gunakan Persamaan (2) dan (3) berikut:

$$\boldsymbol{x} = \mathbf{cos}\left(\boldsymbol{a_n} \times \boldsymbol{d_{obstacle_i}}\right) \quad (2)$$

$$\boldsymbol{y} = \mathbf{sin}\left(\boldsymbol{a_n} \times \boldsymbol{d_{obstacle_i}}\right) \quad (3)$$

Nilai a_n merupakan hasil pembacaan penambahan sudut dari sensor RPLidar dari hasil pemindaian secara 2D . Sedangkan nilai $d_{obstacle_i}$ merupakan hasil pembacaan jangkauan dari sensor RPLidar. Nilai diatas akan menjadikan sudut x dan y sesuai dengan prespektif dari *drone*, yang akan selalu di perbaharui tergantung posisi dari *drone* dengan objek diteksi. Selanjutnya akan dilakukan kalkulasi untuk mengetahui apakah terjadi halangan menggunakan algoritma Bug2 sesuai dengan Persamaan (1).

$$\boldsymbol{x_{avoid}} = \mathbf{0} + \boldsymbol{x} \times \boldsymbol{u} \quad (4)$$

$$\boldsymbol{y_{avoid}} = \mathbf{0} + \boldsymbol{y} \times \boldsymbol{u} \quad (5)$$

Setelah nilai u pada Persamaan (1) berhasil ditemukan maka, akan dilakukan konversi parameter kepada nilai x_{avoid} dan y_{avoid} sebagai nilai koordinat terbaru. Nilai x_{avoid} dan y_{avoid} berasal dari perkalian nilai koordinat dengan nilai algoritma bug2 pada Persamaan (4) dan (5).

$$\boldsymbol{i} = \sqrt{(\boldsymbol{x_{avoid}})^2 + (\boldsymbol{y_{avoid}})^2} \quad (6)$$

Nilai i pada Persamaan (6) merupakan besaran jarak dari halangan terbaru yang di hadapi, nilai jarak ini di dapatkan dari hasil nilai x_{avoid} dan y_{avoid} seperti pada persamaan 5.6. setelah didapatkan nilai i maka, dapat di lakukan seleksi apakah jarak yang di hadapi sudah sesuai dengan parameter yang di tentukan atau tidak. Apabila nilai i sudah memenuhi syarat maka nilai x_{avoid} dan y_{avoid} akan di perbaharui untuk menyesuaikan dengan nilai i yang telah di dapatkan.

$$\boldsymbol{x_{avoid}} = \boldsymbol{e} \times \left(\frac{\boldsymbol{x_{avoid}}}{\boldsymbol{i}}\right) \quad (7)$$

$$\boldsymbol{y_{avoid}} = \boldsymbol{e} \times \left(\frac{\boldsymbol{y_{avoid}}}{\boldsymbol{i}}\right) \quad (8)$$

Nilai e pada Persamaan (7) dan (8) merupakan batas dari jarak halangan yang sudah di tentukan sebelumnya. Setelah nilai x_{avoid} dan y_{avoid} di perbaharui pada perhitungan diatas untuk menyesuaikan dengan kondisi terbaru maka, penambahan dari posisi terbaru sudah di dapatkan. Untuk melakukan penyesuaian dengan koordinat pada *drone* maka di lakukan perhitungan berikut:

$$\boldsymbol{x_t} = \boldsymbol{x_{avoid}} + \boldsymbol{x_{poss}} \quad (9)$$

$$y_t = y_{avoid} + y_{poss} \quad (10)$$

Nilai x_t dan y_t pada Persamaan (9) dan (10) merupakan koordinat x dan y terhadap waktu terbaru yang kemudian akan di kirimkan ke Pixhawk. Nilai tersebut akan selalu di perbaharui menyesuaikan dengan halangan yang di hadapi. Waktu yang digunakan dalah sistem komputasi ini dapat di atur pada parameter *rospy.Rate(x)*, semakin kecil nilai x maka waktu yang dibutuhkan akan semakin lama dan begitu pun sebaliknya apabila nilai x semakin besar maka waktu komputasi akan semakin cepat.

4. PENGUJIAN DAN ANALISIS

4.1 Pengujian Parameter Kecepatan (s) dan Jarak DIteksi (d) Menggunakan Simulator Gazebo

Pada pengujian penggunaan parameter kecepatan (s) dan jarak diteksi (d) pada algoritma Bug2 menggunakan media Simulator Gazebo akan dilakukan secara langsung pada Jetson Nano. Pengujian ini akan dilakukan sebanyak 10 kali dengan penggunaan parameter kecepatan dan jarak diteksi yang berbeda-beda. Tujuan dari pengujian ini adalah mendapatkan nilai dari parameter kecepatan (s) dan jarak diteksi (d) untuk dapat di masukkan pada algoritma Bug2.

Tabel 1. Hasil Pengujian Kecepatan (s) dan Jarak Diteksi (d) pada Simulator Gazebo

s	d	Waktu Tempuh (detik)			Ket.
		Mulai	Akhir	Selisih	
0.5	1	48	76	28	Berhasil
1	1	50	70	20	Berhasil
1.5	1	-	-	-	Gagal
2	1	-	-	-	Gagal
2.5	1	-	-	-	Gagal
0.5	1.5	48	77	29	Berhasil
1	1.5	48	68	20	Berhasil
1.5	1.5	-	-	-	Gagal
2	1.5	-	-	-	Gagal
2.5	1.5	-	-	-	Gagal
0.5	2	48	81	33	Berhasil
1	2	48	71	23	Berhasil
1.5	2	48	69	21	Berhasil
2	2	-	-	-	Gagal
2.5	2	-	-	-	Gagal
0.5	2.5	70	100	30	Berhasi
1	2.5	-	-	-	Gagal
1.5	2.5	-	-	-	Gagal
2	2.5	-	-	-	Gagal
2.5	2.5	-	-	-	Gagal

Pada Tabel 1 merupakan hasil dari pengujian penentuan parameter kecepatan (s) pada *drone* dan jarak diteksi (d) pada sensor RPLidar. Pada hasil pengujian ke 7 dengan kecepatan (s) pada 1m/s dan jarak diteksi (d) pada 1.5m mendapatkan waktu tempuh total pada 20 detik, dengan jarak berhenti dari *drone* pada saat sudah menditeksi halangan tersisa 30 cm. Hasil dari pengujian ke 7 ini merupakan hasil terbaik dari 20 kali pengujian dan pengujian yang berhasil hanya 8 dari 20 kali pengujian.

4.2 Pengujian Akurasi Dari Proses Diteksi Halangan pada *Drone* Menggunakan Algoritma Bug2

Pengujian diteksi halangan pada *drone* menggunakan algoritma Bug 2 akan dilakukan pada *basement* Gedung G Fakultas Ilmu Komputer. Pengujian ini akan dilakukan sebanyak 10 kali dengan halangan satu buah tiang struktur dengan orientasi tiang yang akan diubah pada masing-masing pengujian. Pada pengujian ini *drone* akan diletakkan sejauh 3 meter dari halangan tiang struktur. Pada pengujian ini terdapat keluaran berupa *path* atau jejak jelajah dari *drone* dalam melakukan diteksi halangan menggunakan algoritma Bug2 pada aplikasi RVIZ. Hasil dari pengujian terdapat pada Tabel 2 berikut.

Tabel 2. Hasil Pengujian Akurasi Diteksi Halangan

No	Kesesuaian Jejak	Kesesuaian Titik Akhir
1	Sesuai	Sesuai
2	Sesuai	Sesuai
3	Sesuai	Sesuai
4	Tidak	Tidak
5	Sesuai	Sesuai
6	Sesuai	Sesuai
7	Sesuai	Sesuai
8	Sesuai	Sesuai
9	Sesuai	Sesuai
10	Sesuai	Sesuai

$$Akurasi = 1 - \left|\frac{9}{10} - 1\right| = 90\% \quad (11)$$

Berdasarkan Persamaan 11 diatas akurasi yang didapatkan pada pengujian ini adalah 90% dari total 10 kali pengujian. Dimana 1 dari 9 percobaan terjadi *error* pada hasil pembacaan sensor RPLidar yang mengakibatkan posisi *land* yang tidak sesuai.

4.3 Pengujian Perbandingan Simulator Dengan *Drone* Pada Nilai Kecepatan Dan Waktu Tempuh Dalam Diteksi Halangan Berbasis Algoritma Bug2

Pada pengujian ini akan mendapatkan nilai *error* yang akan menjadi acuan apakah terdapat

kemiripan antara simulator dengan *drone* asli. Pada perbandingan ini kondisi lingkungan halangan pada Gazebo akan dibuat semirip mungkin dengan pada aslinya. Penggunaan parameter kecepatan (s) dan jarak diteksi (d) yang telah didapatkan pada pengujian sebelumnya akan dibuktikan pada aplikasi secara langsung pada *drone* dan simulator Gazebo.

Tabel 3. Hasil Pengujian Waktu Tempuh dan Persentase Penggunaan CPU

Waktu Tempuh (detik)	Persentase Penggunaan CPU (%)
Drone	
24.5	62%
23.9	60%
25.1	62%
25.5	61%
24.7	63%
24.1	64%
23.2	65%
23.6	64%

Dari serangkaian pengujian pada Tabel 3 dengan melakukan perulangan sebanyak 8 kali dengan orientasi halangan yang dilakukan perubahan didapatkan nilai waktu tempuh rata-rata sebesar 24.325 detik. Waktu tempuh yang didapatkan oleh *drone* merupakan waktu yang diambil pada saat *drone* melakukan *takeoff* dan berakhir pada *landing*. Serta sistem komputasi yang tergolong ringan mampu membuat algoritma Bug2 ini menjadi salah satu opsi terbaik untuk melakukan deteksi halang rintang, dengan mendapat rata-rata penggunaan CPU diangka 63% pada Raspberry Pi. Waktu tempuh dan penggunaan CPU yang ringan pada sistem ini sangat memungkinkan untuk menjadi salah satu fitur tambahan pada kontes robot terbang Indonesia. Dari hasil yang telah dilakukan pengujian dapat dijelaskan bahwa *drone* sudah mampu untuk melakukan deteksi halang rintang seraca cepat dengan membaca secara otomatis pada kondisi lingkungannya. Sehingga dapat dikatakan bahwa penelitian ini sudah sesuai dengan latar belakang dan tujuan utama dari penelitian ini.

5. KESIMPULAN DAN SARAN

Berdasarkan seluruh hasil pengujian yang telah dilakukan maka pada hasil dari algoritma Bug 2 memliki hasil diteksi yang cepat dengan waktu diteksi didapatkan dengan waktu 24.325 detik pada sistem *drone* asli dengan akurasi sebesar 90%. Pada penggunaan CPU mendapatkan nilai 63% pada Raspberri PI 4.

Berdasarkan hasil kesimpulan yang telah didapatkan pada penelitian ini, terdapat beberapa masukan atau saran untuk dapat menunjang penelitian-penelitian selanjutnya untuk dapat melakukan pengembangan pada sistem ini secara lebih lanjut, yaitu: Menggunakan Flight Controller yang resmi dari beberapa perusahaan, seperti Pixhawk atau APM, untuk menekan kesalahan yang didapatkan pada Flight Controller, Menggunakan sensor PX4Flow sebagai sensor OpticalFlow untuk mendukung pembacaan ketinggian yang lebih baik dari penelitian ini, Melakukan tuning PID pada Pixhawk pada kondisi luar ruangan dan tanpa angin untuk mendapatkan nilai PID terbaik dan membuat drone lebih stabil saat terbang, Menggunakan pelindung baling-baling untuk menekan kerusakan pada baling-baling pada saat terjadi kecelakaan, Mencoba melakukan explorasi dari algoritma Bug lainnya seperti TangentBug, dan OneBug, Membuat halangan yang lebih bervariasi untuk dapat melakukan pengecekan keseluruhan dari sistem algoritma yang akan dikembangkan.

6. DAFTAR PUSTAKA

Arif, M. *et al.* (2020) 'An Intelligent Obstacle and Edge Recognition System using Bug Algorithm', *Technology, and Sciences (ASRJETS) American Scientific Research Journal for Engineering*, 64(1), pp. 133–143.

Blok, P.M. *et al.* (2019) 'Robot navigation in orchards with localization based on Particle filter and Kalman filter', *Computers and Electronics in Agriculture*, 157, pp. 261–269. Available at: https://doi.org/10.1016/j.compag.2018.12.046.

Divya, S.S. *et al.* (2021) 'An Investigation of Bug Algorithms for Mobile Robot Navigation and Obstacle Avoidance in Two-Dimensional Unknown Static Environments', in *Proceedings - International Conference on Communication, Information and Computing Technology, ICCICT 2021*. Institute of Electrical and Electronics Engineers Inc. Available at: https://doi.org/10.1109/ICCICT50803.2021.9510118.

McGuire, K.N., de Croon, G.C.H.E. and Tuyls, K. (2019) 'A comparative study of bug algorithms for robot navigation', *Robotics and Autonomous Systems*, 121(August). Available at: https://doi.org/10.1016/j.robot.2019.103261.

Mohsen, A.M., Sharkas, M.A. and Zaghlol, M.S. (2019) 'New real time (M-Bug) algorithm for path planning and obstacle avoidance in 2D unknown environment', in *29th International Conference on Computer Theory and Applications, ICCTA 2019 - Proceedings*. Institute of Electrical and Electronics Engineers Inc., pp. 25–31. Available at: https://doi.org/10.1109/ICCTA48790.2019.9478801.

Moysis, L. *et al.* (2020) 'A Chaotic Path Planning Method for 3D Area Coverage Using Modified Logistic Map and a Modulo Tactic', in *2020 International Conference on Unmanned Aircraft Systems, ICUAS 2020*. Institute of Electrical and Electronics Engineers Inc., pp. 220–227. Available at: https://doi.org/10.1109/ICUAS48674.2020.9213954.

Pozna, C. *et al.* (2022) 'Hybrid Particle Filter-Particle Swarm Optimization Algorithm and Application to Fuzzy Controlled Servo Systems', *IEEE Transactions on Fuzzy Systems*, 30(10), pp. 4286–4297. Available at: https://doi.org/10.1109/TFUZZ.2022.3146986.

Ribeiro, M.I. (2005) *Obstacle Avoidance*.

Xu, Q.L., Yu, T. and Bai, J. (2017) 'The mobile robot path planning with motion constraints based on Bug algorithm', *Proceedings - 2017 Chinese Automation Congress, CAC 2017*, 2017-Janua, pp. 2348–2352. Available at: https://doi.org/10.1109/CAC.2017.8243168.

Xuexi, Z. *et al.* (2019) *SLAM Algorithm Analysis of Mobile Robot Based on Lidar; SLAM Algorithm Analysis of Mobile Robot Based on Lidar*.

Yang, S. *et al.* (2019) 'A New Intelligent Trajectory Planning Algorithm Based on Bug2 Algorithm:Bridge Algorithm', in *IEEE International Conference on Robotics and Biomimetics, ROBIO 2019*. Institute of Electrical and Electronics Engineers Inc., pp. 2079–2084. Available at: https://doi.org/10.1109/ROBIO49542.2019.8961390.