Pengembangan Lingkungan Simulasi untuk Pengujian *Socially Assistive Robots* Menggunakan ROS 2 dan Gazebo

1st Muhammad Alfi Maulana Fikri
Departemen Teknik Komputer
Fakultas Teknologi Elektro
dan Informatika Cerdas
Insititut Teknologi Sepuluh Nopember
Surabaya, Indonesia 60111
maulana.17072@mhs.its.ac.id

2nd Mauridhi Hery Purnomo
Departemen Teknik Komputer
Fakultas Teknologi Elektro
dan Informatika Cerdas
Insititut Teknologi Sepuluh Nopember
Surabaya, Indonesia 60111
hery@ee.its.ac.id

3rd I Ketut Eddy Purnama
Departemen Teknik Komputer
Fakultas Teknologi Elektro
dan Informatika Cerdas
Insititut Teknologi Sepuluh Nopember
Surabaya, Indonesia 60111
ketut@te.its.ac.id

4th Muhtadin
Departemen Teknik Komputer
Fakultas Teknologi Elektro
dan Informatika Cerdas
Insititut Teknologi Sepuluh Nopember
Surabaya, Indonesia 60111
muhtadin@te.its.ac.id

***Ringkasan*—Selama beberapa tahun terakhir, robot telah mengalami perkembangan yang cukup signifikan. Salah satu bentuk perkembangan tersebut adalah *socially assistive robots* (SARs) yang mampu memberikan bantuan kepada pengguna dalam bentuk interaksi sosial. Namun, karena sifatnya yang melibatkan interaksi langsung dengan pengguna, pengujian pada SARs akan menjadi sulit dan beresiko. Untuk itu, pada penelitian ini kami mengajukan lingkungan simulasi untuk pengujian SARs yang dibuat menggunakan simulator Gazebo. Di dalam lingkungan simulasi ini, model robot akan diujikan dengan model pengguna serta model-model objek lain secara virtual. Agar pengujian yang dilakukan di simulasi bisa diterapkan pada *real robot*, sistem kontroler yang ada pada robot akan dibuat secara terabstraksi dengan memisah setiap komponen menjadi *nodes* menggunakan ROS 2. Hasilnya, model pengguna dan ruangan virtual di simulasi mampu digunakan dalam percobaan deteksi pose dan percobaan SLAM. Sistem yang dibuat tersebut mampu menghasilkan tindakan yang sama dalam menggerakkan model robot dan *real robot* dengan perbedaan *error* sebesar 2.6% di simulasi dan 12.5% di dunia nyata. Lebih lanjut, Dengan performa yang dimiliki ROS 2, pengiriman citra dengan resolusi hingga 640 x 480 mampu menghasilkan *delay* di bawah 50 ms dan frekuensi di atas 90% pada sesama perangkat maupun antar-perangkat baik di simulasi maupu di dunia nyata.**

***Kata kunci*—Simulasi, *Assistive Robotics*, ROS2, Gazebo.**

I. Pendahuluan

Selama beberapa tahun terakhir, robot telah mengalami perkembangan yang signifikan dari robot beroda untuk edukasi [1] hingga robot *manipulator* untuk skala industri [2]. Salah satu bentuk perkembangan lain dari robot tersebut adalah *socially assistive robots* (SARs). SARs merupakan jenis robot dalam bidang *socially assistive robotics* yang menggabungkan aspek yang ada pada *assistive robotics* dan *socially interactive robotics* sehingga menjadikan SARs sebagai robot yang mampu memberikan bantuan kepada pengguna dalam bentuk interaksi sosial [3].

Namun, karena sifat dari SARs yang melibatkan interaksi langsung dengan pengguna, maka pengujian dari robot akan menjadi sulit dan beresiko bagi pengguna yang ikut terlibat dalam pengujian tersebut [4]. Salah satu solusi untuk mengatasi masalah tersebut adalah dengan melakukan pengujian secara virtual melalui simulasi robot. Selain bisa meminimalisir resiko, penggunaan simulasi robot sebagai media pengujian robot juga bisa mengurangi biaya yang dibutuhkan dan menghemat waktu pengujian selama pengembangan robot tersebut [5].

Hingga saat ini sudah ada beberapa simulator yang bisa digunakan untuk menjalankan simulasi robot seperti Webots [6], Gazebo [7], V-REP [8], OpenAI Gym [9], dan lain sebagainya. Namun, simulator-simulator tersebut hanyalah *platform* yang secara umum digunakan untuk membantu pengembangan robot melalui simulasi virtual. Sedangkan pengembangan dari lingkungan simulasi dan kontroler robot untuk simulasi tersebut harus dibuat sendiri oleh pengembang robot.

Untuk itu, pada penelitian ini kami mengajukan penelitian terkait pengembangan lingkungan simulasi untuk pengujian SARs menggunakan ROS 2 dan Gazebo. ROS 2 dan Gazebo sendiri dipilih karena tersedianya banyak *library* yang dapat membantu pengembangan maupun pengujian robot, terutama untuk simulasi. Selain itu, dengan adanya skema *hardware abstraction* pada ROS 2, kontroler robot yang diuji melalui simulasi bisa dengan mudah dipindahkan ke robot fisik untuk diuji secara langsung pada pengguna [5].

II. Penelitian Terkait

Beberapa penelitian sebelumnya telah berhasil dalam mengembangkan lingkungan simulasi untuk robot menggunakan ROS (Pendahulu ROS 2) dan Gazebo. Seperti yang dilakukan Qian et al. [10] yang mengembangkan simulasi untuk robot *manipulator*, Zhang et al. [11] yang mengembangkan simulasi untuk robot *quadrotor UAV*, dan Takaya et al. [5] yang mengembangkan lingkungan simulasi untuk pengujian terhadap *mobile robot*. Namun, berbeda dengan penelitian yang telah dilakukan sebelumnya, penelitian yang akan kami lakukan memilih menggunakan ROS 2 agar kontroler robot yang dibuat untuk simulasi memiliki performa yang lebih baik serta dapat bekerja secara *real-time* [12].

Selain itu, penelitian lain juga telah dilakukan oleh Erickson et al. [4] yang mengembangkan Assistive Gym, sebuah *framework* simulasi untuk *assistive robotics* berbasis OpenAI Gym. *Framework* simulasi tersebut kemudian digunakan oleh Clegg et al. [13] untuk mengembangkan metode *learning* melalui simulasi pada kolaborasi antara robot dengan manusia dalam membantu pemakaian baju pada manusia. Namun, karena tidak menggunakan ROS, kontroler robot yang dibuat untuk simulasi yang menggunakan *framework* tersebut perlu dibuat ulang ketika akan diujikan secara langsung pada pengguna menggunakan robot fisik. Walaupun begitu, penelitian yang dilakukan oleh Zamora et al. [14] menunjukkan bahwa simulasi yang ada pada OpenAI Gym juga bisa diintegrasikan pada ROS dan Gazebo, sehingga tidak menutup kemungkinan bahwa Assistive Gym juga bisa digunakan bersamaan dengan ROS 2 dan Gazebo.

III. ROS 2 dan Gazebo

A. Robot Operating System 2 (ROS 2)

Robot Operating System (ROS) [15] merupakan kumpulan dari *libraries*, *drivers*, dan *tools* yang mempermudah pengembangan sistem pada robot. ROS memiliki *command tool* seperti Linux, sistem komunikasi antar proses, dan berbagai macam *packages* yang berhubungan dengan pengembangan sistem pada robot.

Proses yang dieksekusi pada ROS disebut sebagai *node*, komunikasi antar proses yang dimiliki menggunakan model *publish/subscribe*, dan data komunikasi yang dikirimkan disebut sebagai *topic*. Suatu proses *publisher* mampu mengirimkan satu maupun lebih *topic*, kemudian proses-proses lain yang melakukan *subscribe* pada suatu *topic* bisa memperoleh isi dari *topic* tersebut. Selain itu ada juga *sevice* yang memiliki fungsi seperti *topic*, hanya saja dilakukan secara dua arah. *Service* ini bekerja menggunakan model *client/server* dimana *service client* akan mengirimkan data permintaan dalam bentuk *request* dan kemudian *service server* akan mengirimkan data balasan dalam bentuk *response*.

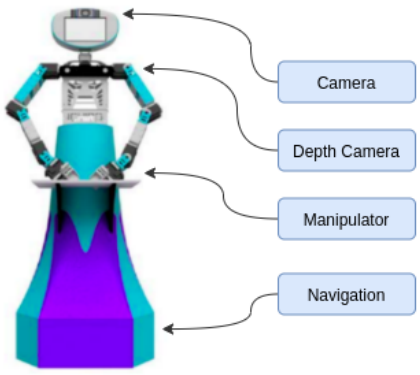
Generasi kedua dari Robot Operating System, ROS 2, merupakan kelanjutan dari ROS yang mengusung reliabilitas dan performa untuk penggunaan *real-time* sembari masih mendukung keunggulan yang dimiliki oleh ROS sebelumnya [12]. Untuk memenuhi kebutuhan reliabilitas dan performa untuk penggunaan real-time tersebut, ROS 2 menggunakan *Data Distribution Service* (DDS) [16][17], standar industri untuk sistem komunikasi *real-time* dan *end-to-end middleware*, yang menggantikan sistem komunikasi antar proses yang dimiliki ROS sebelumnya.

B. Gazebo

Gazebo [7] merupakan bagian dari Player Project [18] yang memungkinkan sebuah simulasi robot dan aplikasi sensor bekerja di lingkungan simulasi indoor maupun outdoor tiga dimensi. Gazebo memiliki arsitektur *client/server* dan model *publish/subscribe* untuk sistem komunikasi antar prosesnya. Setiap objek simulasi di Gazebo dapat diasosiasikan dalam satu maupun lebih kontroler yang akan memproses perintah untuk mengatur dan menentukan keadaan dari suatu objek. Data yang dihasilkan oleh suatu kontroler akan dikirim ke *shared memory* menggunakan *Gazebo interfaces* (*ifaces*). Nantinya *ifaces* dari proses-proses lain dapat membaca data tersebut pada *shared memory*, sehingga memungkinkan komunikasi antar proses antara program yang mengontrol robot dan Gazebo, terlepas dari bahasa pemrograman yang digunakan.

IV. Desain Sistem

A. Desain Robot

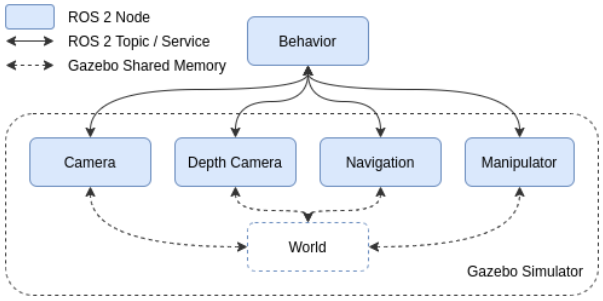


Gambar 1. Diagram komponen yang ada pada robot *Dienen*.

Robot yang akan digunakan pada pada penelitian ini adalah robot *Dienen* yang merupakan kelanjutan dari robot *IRIS* [19][20] dengan penambahan desain dari robot *ICHIRO* [21] di bagian atas robot. Desain seperti ini secara umum dikenal sebagai desain *mobile humanoid robot* [22], yang merupakan desain gabungan antara robot *mobile* dan robot *humanoid*. Seperti yang terlihat pada gambar 1, bagian bawah robot menyerupai robot *mobile* dengan tiga buah penggerak *omnidirectional wheels* yang memungkinkan pergerakan robot secara *holonomic* [23], sedangkan bagian atas robot menyerupai robot *humanoid* yang terdiri atas badan, kepala, dan lengan. Dengan desain *mobile humanoid robot* ini, diharapkan pengguna bisa merasakan interaksi sosial yang lebih baik dengan robot karena memiliki bentuk mendekati manusia [24] sekaligus mempermudah navigasi dari robot ke berbagai tempat.

Robot *Dienen* dilengkapi dengan beberapa sensor seperti IMU (*inertial measurement unit*) untuk mengetahui orientasi dari robot, *rotary encoder* untuk melakukan perhitungan odometri dari robot, *distance sensor* untuk mendeteksi adanya objek lain di sekitar robot, sensor kamera di kepala untuk menangkap citra, dan sensor *depth camera* yang nantinya bisa digunakan untuk melakukan pemetaan dari ruangan. Selain itu robot ini juga dilengkapi dengan dua lengan seperti robot *manipulator* yang bisa diatur pada berbagai posisi dan orientasi [25]. Dengan adanya sensor dan aktuator ini, diharapkan robot mampu melakukan tindakan *assistive* secara sosial sesuai dengan data yang didapatkan dari sensor yang ada.

B. Desain Kontroler Robot

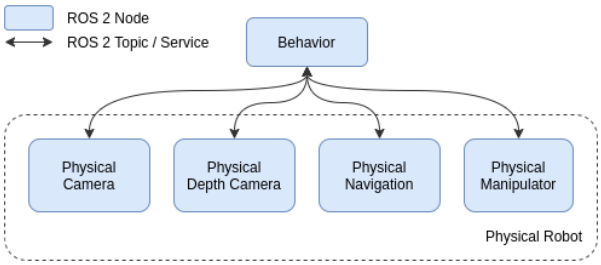


Gambar 2. Diagram sistem kontroler robot di simulasi dengan pemisahan *behavior* dari sensor dan aktuator virtual.

Kontroler robot yang digunakan untuk simulasi ini akan dikembangkan menggunakan ROS 2. Kontroler tersebut akan dipisah menjadi beberapa bagian dalam bentuk *ROS 2 node* seperti yang terlihat pada gambar 2. Setiap *node* yang ada akan terhubung satu sama lain menggunakan sistem komunikasi antar proses ROS 2 yang berupa *topics* dan *services*.

Bagian utama dari kontroler robot tersebut adalah *node behavior* yang berisi program yang mengatur segala tindakan robot berdasarkan data yang didapat dari sensor yang ada di simulasi. Kemudian *node behavior* tersebut akan terhubung dengan empat *node* lain yang merepresentasikan sensor dan aktuator yang ada pada robot. Keempat *node* tersebut akan terpasang di dalam cakupan simulator Gazebo sebagai *Gazebo plugins*, sehingga mampu digunakan untuk mengakses dan memanipulasi data yang ada di simulasi menggunakan sistem *shared memory* pada Gazebo [26].

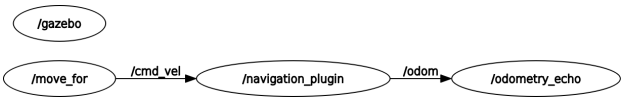
Sistem ini dirancang secara terpisah agar *node behavior* yang diujikan di lingkungan simulasi bisa langsung bekerja pada robot fisik. Seperti yang terlihat pada gambar 3, transfer kontroler ke robot fisik dapat dilakukan dengan mengubah keseluruhan cakupan yang ada di simulator Gazebo, yang berupa empat *node* yang telah disebutkan sebelumnya, menjadi *node* yang memproses sensor dan aktuator yang ada pada robot fisik. Dengan ini pengujian yang dilakukan di simulasi bisa langsung diterapkan ketika diujikan pada robot fisik karena tidak perlunya pembuatan ulang kontroler yang menyesuaikan sistem yang ada pada robot.



Gambar 3. Diagram sistem kontroler robot untuk robot fisik dengan pemisahan *behavior* dari perangkat sensor dan aktuator yang dimiliki robot.

V. Hasil Pengujian

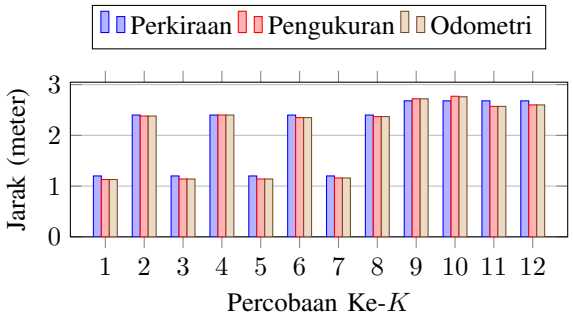
A. Pengujian Gerakan



Gambar 4. Skema *node* dari pengujian gerakan di simulasi.

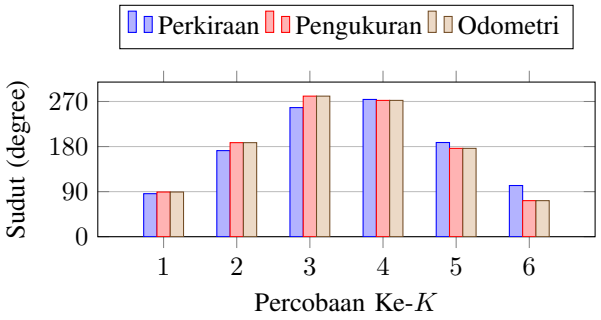
Gambar 5. Skema *node* dari pengujian gerakan pada robot fisik.

Pengujian gerakan dilakukan dengan menjalankan *node* `move_for` sebagai *node behavior* yang akan memerintahkan robot untuk bergerak dengan kecepatan tertentu selama kurun waktu tertentu. Seperti yang terlihat pada gambar 4, di simulasi, *node* `move_for` akan terhubung dengan *node* `navigation_plugin` untuk mengatur kecepatan dari robot yang ada di simulasi menggunakan *topic* `/navigation/maneuver_input`. Sedangkan untuk pengujian di dunia nyata, seperti yang terlihat pada gambar 5, peran dari *node* `navigation_plugin` yang mengatur navigasi pada robot virtual akan digantikan oleh *node* `navigation` yang mengatur navigasi yang ada pada robot fisik.



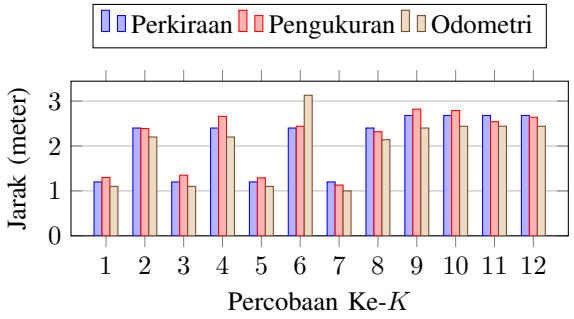
Gambar 6. Perbandingan hasil jarak akhir dari nilai yang diharapkan dan yang terukur di simulasi.

Pengujian gerakan terbagi menjadi dua bagian, pengujian gerakan linier dan pengujian gerakan putar, masing-masing diujikan dengan berbagai macam kombinasi nilai kecepatan selama 3 detik pada robot virtual di lingkungan simulasi dan

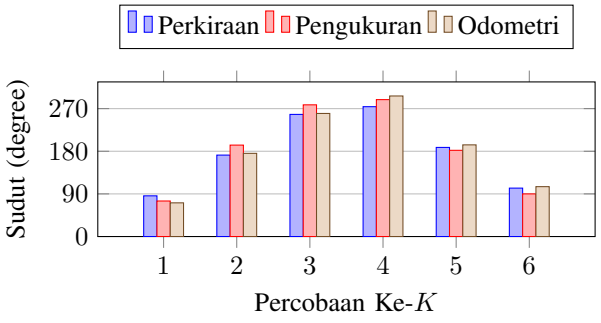


Gambar 7. Perbandingan hasil orientasi akhir dari nilai yang diharapkan dan yang terukur di simulasi.

pada robot fisik di dunia nyata. Seperti yang terlihat pada gambar 6 dan gambar 7, jarak dan orientasi akhir odometri yang diterima robot memiliki nilai yang relatif sama dengan jarak dan orientasi akhir dari nilai perkiraan dari kecepatan yang diberikan dan pengukuran dari titik awal model robot di simulasi. Dari hasil tersebut, persentase *error* dari nilai odometri dibandingkan rata-rata nilai perkiraan dan pengukuran di simulasi adalah sebesar 2.6%.



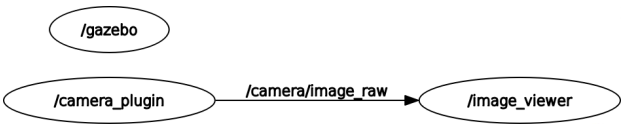
Gambar 8. Perbandingan hasil jarak akhir dari nilai yang diharapkan dan yang terukur pada robot fisik.



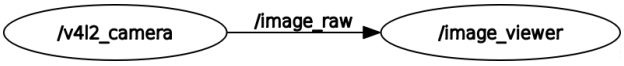
Gambar 9. Perbandingan hasil orientasi akhir dari nilai yang diharapkan dan yang terukur pada robot fisik.

Berbeda dengan hasil pengujian di lingkungan simulasi, hasil pengujian di dunia nyata memiliki sedikit perbedaan antara nilai perkiraan dan pengukuran dengan nilai odometri yang diterima oleh robot. Seperti yang terlihat pada gambar 8 dan gambar 9, terdapat sedikit perbedaan di antara keduanya yang disebabkan oleh kombinasi berbagai faktor seperti ketidakakuratan pembacaan sensor dan terjadinya *slip* ketika robot bergerak dengan akselerasi yang tinggi. Karena adanya faktor tersebut, persentase *error* dari nilai odometri dibandingkan rata-rata nilai perkiraan dan pengukuran pada robot fisik adalah sebesar 12.5%. Walaupun begitu, terlepas dari adanya perbedaan tingkat akurasi, dengan menggunakan *node behavior* yang sama, robot tergolong mampu melakukan perintah gerakan yang sesuai ketika diujikan di simulasi maupun di dunia nyata.

B. Pengujian Pengiriman Citra

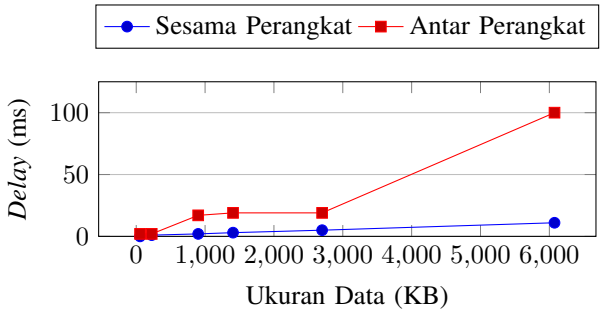


Gambar 10. Skema *node* dari pengujian pengiriman citra di simulasi.



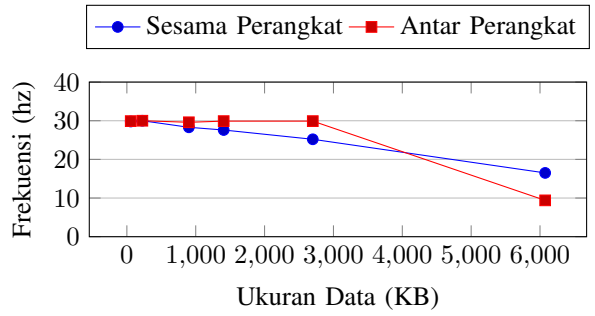
Gambar 11. Skema *node* dari pengujian pengiriman citra pada robot fisik.

Pengujian pengiriman citra dilakukan dengan menjalankan *node* `image_viewer` yang akan menampilkan citra yang diterima dalam bentuk GUI. Seperti yang terlihat pada gambar 10, di simulasi, *node* `image_viewer` akan terhubung dengan *node* `camera_plugin` yang akan mengirimkan data citra melalui *topic* `/camera/image_raw`. Sedangkan untuk pengujian di dunia nyata, seperti yang terlihat pada gambar 11, peran dari *node* `camera_plugin` akan digantikan oleh *node* `v4l2_camera` yang akan mengirimkan data citra yang berasal dari device camera yang ada pada robot fisik.



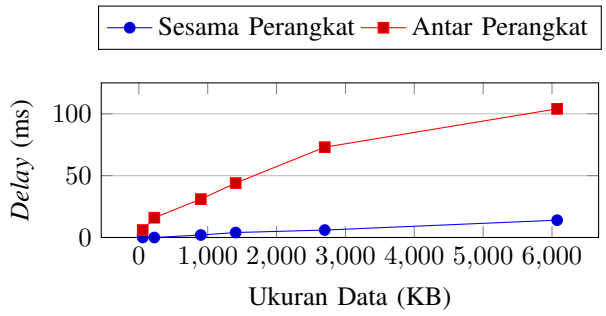
Gambar 12. Hasil *delay* dibandingkan ukuran data dari pengiriman citra di simulasi.

Pengujian pengiriman citra dilakukan pada kondisi pengiriman pada perangkat yang sama dan pada pengiriman antara dua perangkat yang berbeda. Pengujian tersebut masing-masing dilakukan dengan berbagai macam kombinasi nilai resolusi dari 160x120 hingga 1920x1080 pada robot virtual di lingkungan simulasi maupun pada robot fisik di dunia nyata. Seperti yang terlihat pada gambar 12 dan gambar

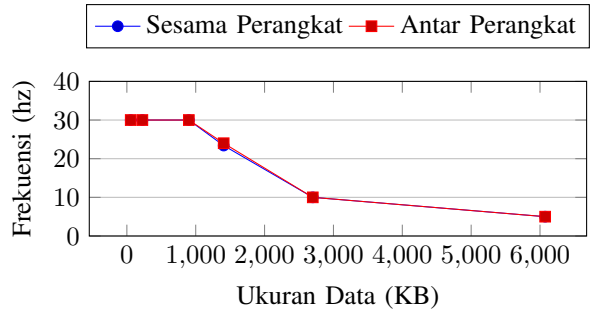


Gambar 13. Hasil frekuensi dibandingkan ukuran data dari pengiriman citra di simulasi.

13, di simulasi, nilai *delay* yang dihasilkan cenderung naik sedangkan nilai frekuensi yang dihasilkan cenderung turun seiring dengan besarnya data yang dikirimkan. Dari hasil tersebut, hingga resolusi 800x600 ($\pm$2500 KB), pengiriman citra di simulasi mampu menghasilkan *delay* rendah di bawah 50 ms dan frekuensi stabil di atas 90% dari nilai normalnya.



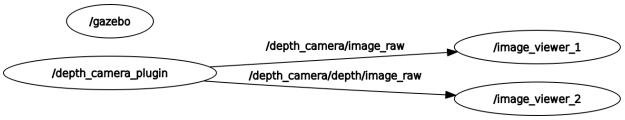
Gambar 14. Hasil *delay* dibandingkan ukuran data dari pengiriman citra pada robot fisik.



Gambar 15. Hasil frekuensi dibandingkan ukuran data dari pengiriman citra pada robot fisik.

Sedangkan, pada robot fisik, seperti yang terlihat pada gambar 14 dan gambar 15, hasil yang didapatkan juga relatif sama dimana nilai *delay* cenderung naik sedangkan nilai frekuensi cenderung turun. Namun, berbeda dengan hasil pengujian di lingkungan simulasi, dari hasil pada robot fisik tersebut, didapatkan bahwa hingga resolusi 640x480 ($\pm$1500 KB), pengiriman citra pada robot real mampu menghasilkan *delay* rendah di bawah 50 ms dan frekuensi stabil di atas 90% dari nilai normalnya.

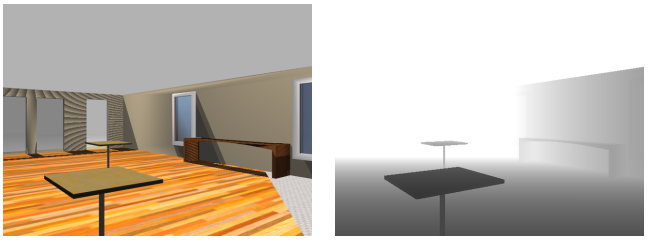
C. *Pengujian* depth camera



Gambar 16. Skema *node* dari pengujian *depth camera* di simulasi.

Gambar 17. Skema *node* dari pengujian *depth camera* pada robot fisik.

Pengujian *depth camera* dilakukan untuk menguji kemampuan *depth camera* di simulasi dalam mensimulasikan perangkat *depth camera*, sekaligus menguji abstraksi perangkat yang ada pada ROS 2. Pengujian ini dilakukan dengan menjalankan dua buah *node* `image_viewer` yang masing-masing akan menerima data citra warna dan citra kedalaman. Seperti yang terlihat pada gambar 16, di simulasi, masing-masing *node* `image_viewer` akan menerima data citra warna melalui *topic* `/depth_camera/image_raw` dan data citra kedalaman melalui *topic* `/depth_camera/depth/image_raw`. Sedangkan untuk pengujian di dunia nyata, seperti yang terlihat pada gambar 17, peran dari *node* `depth_camera_plugin` akan digantikan oleh *node* `v4l2_camera` yang akan mengirimkan data yang berasal dari perangkat Kinect V2 yang ada pada robot fisik.



(a) Citra warna (b) Citra kedalaman

Gambar 18. Perbandingan hasil tangkapan citra warna dan citra kedalaman dari *depth camera* di simulasi.

Hasilnya, seperti yang terlihat pada gambar 18, *node* `image_viewer` menampilkan data yang diterima dari simulasi dimana gambar kiri (18a) menunjukkan citra dengan tampilan berwarna, sedangkan gambar kanan (18b) menunjukkan citra kedalaman dengan tampilan hitam putih. Pada citra kedalaman, terangnya suatu titik pada citra tersebut menunjukkan posisi yang semakin jauh dari jangkauan titik pusat kamera. Sebagai contoh, gelapnya bentuk meja pada citra tersebut menunjukkan posisi objek tersebut lebih dekat dibandingkan objek yang lain.

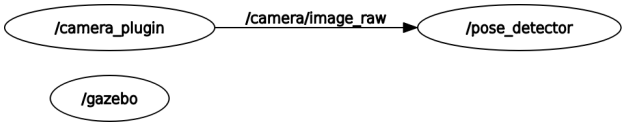
Sedangkan, pada robot fisik hasil yang didapatkan juga relatif sama. Seperti yang terlihat pada gambar 19, gambar kiri

(a) Citra warna (b) Citra kedalaman

Gambar 19. Perbandingan hasil tangkapan citra warna dan citra kedalaman dari *depth camera* pada robot fisik.

(19a) menunjukkan citra warna sedangkan gambar kanan (19b) menunjukkan citra kedalaman. Namun, pada gambar tersebut, citra kedalaman memiliki hasil yang lebih terang, hal ini terjadi karena jangkauan perangkat Kinect V2 yang lebih dekat (4.5 meter) dibandingkan jangkauan yang diatur pada *depth camera* yang ada di simulasi.

D. Pengujian Deteksi Pose



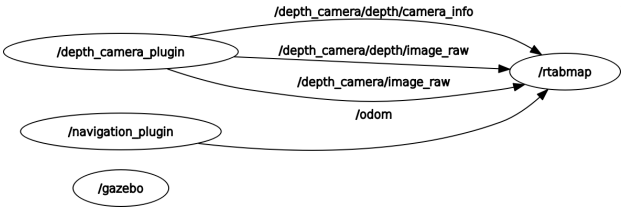
Gambar 20. Skema *node* dari pengujian deteksi pose.

Pengujian deteksi pose dilakukan di simulasi dengan tujuan untuk menguji kemampuan model pengguna dalam mensimulasikan pengguna *real*. Pada pengujian ini, *node* `pose_detector` akan dijalankan untuk melakukan proses visi komputer dalam mendeteksi pose pengguna. Seperti yang terlihat pada gambar 20, *node* `pose_detector` akan terhubung dengan *node* `camera_plugin` yang akan mengirimkan data citra pengguna yang kemudian akan diolah oleh *node* `pose detector` untuk menghasilkan pose yang terdeteksi.

Gambar 21. Hasil deteksi pose pengguna.

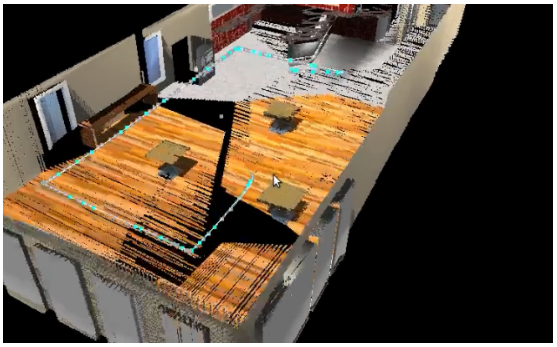
Hasilnya, seperti yang terlihat pada gambar 21, dari citra yang diterima, *node* `pose_detector` akan mendeteksi pose dari pengguna tersebut dan menampilkan hasilnya sebagai garis dan titik pada citra yang diterima. Selain itu, deteksi pose juga mampu menampilkan hasil yang sesuai ketika postur pengguna diubah dari berdiri ke duduk seperti hasil gambar 21 tersebut.

E. Pengujian SLAM



Gambar 22. Skema *node* dari pengujian SLAM.

Pengujian SLAM (*simultaneous localization and mapping*) dilakukan di simulasi dengan tujuan untuk menguji kemampuan ruangan virtual dalam mensimulasikan ruangan *real*. Pada pengujian ini, model robot akan bergerak mengelilingi ruangan sambil melakukan pemetaan menggunakan metode SLAM. Seperti yang terlihat pada gambar 22, *node* `rtabmap` akan terhubung dengan *node* `depth_camera_plugin` dan *node* `navigation_plugin` yang akan mengirimkan data citra warna, citra kedalaman, serta odometri yang akan digunakan dalam proses pemetaan ruangan.



Gambar 23. Hasil pemetaan ruangan.

Hasilnya, seperti yang terlihat pada gambar 23, proses pemetaan ruangan menghasilkan peta tiga dimensi yang tersusun atas *point cloud* yang membentuk ruangan yang diujikan di simulasi. Peta tersebut ditampilkan pada GUI yang dimiliki oleh *node* `rtabmap`. Selain itu, pada peta tersebut juga tampak visualisasi dari lintasan yang telah dilalui robot sebagai garis dengan titik berwarna biru.

VI. Kesimpulan

Dari pembahasan yang telah dijelaskan di bagian sebelumnya, didapatkan bahwa dengan menggunakan sistem yang dirumuskan, sebuah robot, terutama SARs, dapat dikembangkan dan diujikan baik di lingkungan simulasi menggunakan robot virtual maupun di dunia nyata menggunakan robot fisik. Dengan ini, pengembangan robot akan lebih mudah untuk dilakukan, lebih hemat biaya, serta lebih aman untuk diujikan

kepada pengguna karena dapat dikembangkan secara virtual di lingkungan simulasi, sembari tidak meninggalkan kemampuan untuk diujicobakan juga pada robot fisik.

PUSTAKA

[1] P. Gonçalves, P. Torres, C. Alves, F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J. Zufferey, D. Floreano, and A. Martinoli, "The e-puck, a robot designed for education in engineering," *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, vol. 1, 2009.

[2] M. Blatnický, J. Dižo, J. Gerlici, M. Sága, T. Lack, and E. Kuba, "Design of a robotic manipulator for handling products of automotive industry," *International Journal of Advanced Robotic Systems*, vol. 17, no. 1, 2020.

[3] D. Feil-Seifer and M. J. Mataric, "Defining socially assistive robotics," in *9th International Conference on Rehabilitation Robotics, 2005. ICORR 2005.*, 2005, pp. 465–468.

[4] Z. Erickson, V. Gangaram, A. Kapusta, C. K. Liu, and C. C. Kemp, "Assistive gym: A physics simulation framework for assistive robotics," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 10 169–10 176.

[5] K. Takaya, T. Asai, V. Kroumov, and F. Smarandache, "Simulation environment for mobile robots testing using ros and gazebo," in *2016 20th International Conference on System Theory, Control and Computing (ICSTCC)*, 2016, pp. 96–101.

[6] O. Michel, "Cyberbotics ltd. webots™: Professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, p. 5, 2004.

[7] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.

[8] E. Rohmer, S. P. N. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1321–1326.

[9] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[10] W. Qian, Z. Xia, J. Xiong, Y. Gan, Y. Guo, S. Weng, H. Deng, Y. Hu, and J. Zhang, "Manipulation task simulation using ros and gazebo," in *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, 2014, pp. 2594–2598.

[11] M. Zhang, M. Lan, J. Lin, S. Wang, K. Liu, F. Lin, and B. M. Chen, "A high fidelity simulator for a quadrotor uav using ros and gazebo," in *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*, 2015, pp. 002 846–002 851.

[12] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ros2," in *2016 International Conference on Embedded Software (EMSOFT)*, 2016, pp. 1–10.

[13] A. Clegg, Z. Erickson, P. Grady, G. Turk, C. C. Kemp, and C. K. Liu, "Learning to collaborate from simulation for robot-assisted dressing," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2746–2753, 2020.

[14] I. Zamora, N. Lopez, V. Vilches, and A. Cordero, "Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo," *arXiv preprint arXiv:1608.05742*, 2016.

[15] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," *ICRA Workshop on Open Source Software*, vol. 3, 2009.

[16] G. Pardo-Castellote, "Omg data-distribution service: architectural overview," in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, 2003, pp. 200–206.

[17] J. M. Schlesselman, G. Pardo-Castellote, and B. Farabaugh, "Omg data-distribution service (dds): architectural update," in *IEEE MILCOM 2004. Military Communications Conference, 2004.*, vol. 2, 2004, pp. 961–967 Vol. 2.

[18] B. Gerkey, R. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," *Proceedings of the International Conference on Advanced Robotics*, 2003.

[19] R. Dikairono, Setiawardhana, D. Purwanto, and T. A. Sardjono, "Cnn-based self localization using visual modelling of a gyrocompass line mark and omni-vision image for a wheeled soccer robot application," *International Journal of Intelligent Engineering and Systems*, 2020.

[20] Muhtadin, R. M. Zanuar, I. K. E. Purnama, and M. H. Purnomo, "Autonomous navigation and obstacle avoidance for service robot," in *2019 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM)*, 2019, pp. 1–8.

[21] M. Muhtadin, M. Arrazi, S. Ali, T. Pratama, D. Wicaksono, A. Putra, I. Ari, A. Maulana, O. Bramastyo, S. Asshakina, M. Attamimi, M. Arifin, M. Hery Purnomo, and D. Purwanto, "Ichiro robots winning robocup 2018 humanoid teensize soccer competitions," in *RoboCup 2018: Robot World Cup XXII*, 2019, pp. 425–435.

[22] Z. Mohamed and G. Capi, "Development of a new mobile humanoid robot for assisting elderly people," *Procedia Engineering*, vol. 41, pp. 345 – 351, 2012.

[23] H. Oliveira, A. Sousa, A. Moreira, and P. Costa, "Dynamical models for omni-directional robots with 3 and 4 wheels," *ICINCO 2008 - Proceedings of the 5th International Conference on Informatics in Control, Automation and Robotics*, vol. 1, 2008.

[24] S. Rossi, M. Staffa, and A. Tamburro, "Socially assistive robot for providing recommendations: Comparing a humanoid robot with a mobile application," *International Journal of Social Robotics*, 2018.

[25] J. Iqbal, M. Ul Islam, and H. Khan, "Modeling and analysis of a 6 dof robotic arm manipulator," *Canadian Journal on Electrical and Electronics Engineering*, vol. 3, pp. 300–306, 2012.

[26] "Gazebo plugins," 2021. [Online]. Available: http://gazebosim.org/tutorials?tut=ros_gzplugins