Dodging Dynamical Obstacles Using Turtlebot4 Camera Feed

Wei-Teng Chu

Abstract The research aims to search for a method for Turtlebot4, a driving robot, to bypass a dynamical obstacle based on an affordable camera while it drives toward some assigned destination. The method of obstacle dodging implemented in this research consists of obstacle detection based on a stereo camera, path planning, and motion control. Most previous research focused on using state-of-the-art instruments such as event cameras or LiDARs to perform an obstacle-dodging function. If the dynamical obstacles dodging technology proposed in this research could be implemented on Turtlebot4, the cost of implementing the functionality of obstacle dodging would be lowered.

Keywords Robotics · Stereo camera · Dynamical obstacles · Obstacles dodging · Control · Deep learning

1 Introduction

Previous research has been conducted concerning dynamic obstacle avoidance within the realm of robotics. For example, [1] demonstrates how a quadrotor could dodge obstacles by making use of an event camera. Additionally, [2] combines the data from the camera and 2D LiDAR for self-driving automobiles to dodge obstacles, which ensures the safety of self-driving automobiles. Furthermore, [3] showcases how humanoid robots could circumvent obstacles by combining vision-based sensing with a footstep planner. However, two of the previous studies I mentioned rely on unaffordable event cameras or LiDARs. One of them involves implementing obstacle avoidance procedures on humanoid robots. Instead of using event cameras or LiDARs, if the vision-based sensing method founded on [4, 5], and [6] could be implemented on the Turtlebot4, it could become fundamental for future research on the cost-lowering of self-driving automobiles.

W.-T. Chu (✉)
College of Engineering, National Tsing Hua University, Hsinchu, Taiwan R.O.C. 300044
e-mail: s109030015@m109.nthu.edu.tw

J. P. T. Mo (ed.), *Proceedings of the 10th International Conference on Mechanical, Automotive and Materials Engineering*, Lecture Notes in Mechanical Engineering,
https://doi.org/10.1007/978-981-97-4806-8_17

2 Preliminaries

Robot Operating System 2, also known as ROS2 provides a malleable architecture for robot engineers to facilitate communication, real-time control, as well as collaboration between various robotic system components. This makes it possible for developers to create, simulate, and deploy robotic applications across a wide range of platforms. The most frequently used structure of ROS2 in my research is "nodes and topics." The robot system and be broken into several nodes, through which the robot can perform specific tasks. The one who releases the data is the "Publisher Node." On the other hand, the one who receives the data is the "Subscriber Node." The data is not transferred directly between them. The publisher needs first to send the message to the specific "Topic." The subscriber, who subscribes to that topic, can then receive the data from that topic. "The Turtlebot4" is used for the dodging robot in my research. It is a ROS2-based mobile robot. It is capable of mapping the robot's surroundings, navigating autonomously, and running Artificial Intelligence models on its camera.

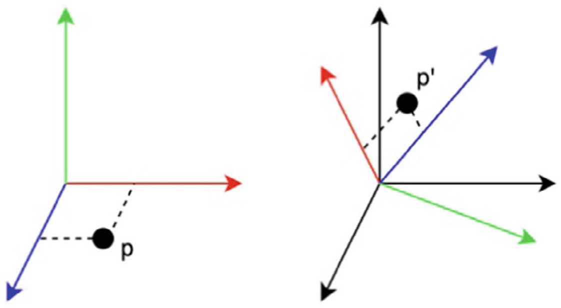
3 Development

3.1 OAK-D Pro Camera

OAK-D Pro is a stereo camera mounted on the Turtlebot4, through which we can get the "depth" information of the object in front of it. In other words, it can provide the distance between the robot and the obstacle. Together with the example code and pre-trained model provided by Luxonis [7], we can track the obstacle and get the coordinate in the camera's frame. The camera can track a person and also provide the coordinates of the person in its frame. However, the robot doesn't know where the obstacle actually is, since the coordinates of the robot itself are different from that of the camera. Thus, we need to deal with some forward kinematics here.

3.2 Forward Kinematics

According to [8, p. 28], the forward kinematics challenge for a serial-chain manipulator involves determining the end-effector's position and orientation concerning the base. This is achieved by utilizing joint positions and geometric link parameter values across all joints. As shown in Fig. 1, if we need to represent the coordinates p in terms of the dashed coordinates system, all we need to do is to find a special matrix that represents the relationship between the two systems. When we multiply p with the matrix, we can get the coordinates of p in terms of the dashed system, which is p'.



$$p = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad p' = Rp = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Fig. 1 An example of transformation. $p = (x, y, z)$ is the coordinates of the point P in the coordinate system on the left side. If we would like to express the coordinates of this point with respect to the other coordinate system shown on the right, a transformation matrix R is needed

Fortunately, there is already information regarding the transformation between the camera frame and the odometry frame stored in two topics in the robot (`/tf` and `/tf_static`). All we need to do is simply subscribe to the topics and retrieve the information. However, the transformation between the two systems is not as simple as the example in the previous paragraph. It is composed of six transformations in total. As shown in the top side of Fig. 2, the transformation starts from "`oakd_rgb_camera_op-tical_frame`" through "`odom`."

All the frames mentioned above and their locations are shown on the bottom side of Fig. 2. Again, the goal is to transform the coordinates in the camera frame into the odometry frame. The camera frame is actually "`oakd_rgb_camera_optical-_frame`" in the figure. (Since the three frames that start with "`oakd`" are too close together, only one of them is shown.) As mentioned before, we can get the transformation information from the topic. What we are going to get are "Translation Data" and a set of "Quaternions." Here, "Quaternions" are an alternate way to describe orientation or rotations in 3D space using an ordered set of four numbers. We can then create a rotational matrix by the expression defined in Eq. 1, where q_0, q_1, q_2, and q_3 are the quaternions.

$$R(Q) = \begin{bmatrix} 2(q_0^2 + q_1^2) - 1 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & 2(q_0^2 + q_2^2) - 1 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & 2(q_0^2 + q_3^2) - 1 \end{bmatrix} \quad (1)$$

After that, we put the rotational matrix into the upper left corner of a 4×4 matrix with translation data on the right side of it. Below the rotational matrix, we put a $(0, 0, 0, 1)$ row vector, as shown in Eq. 2. By doing so, we can create a

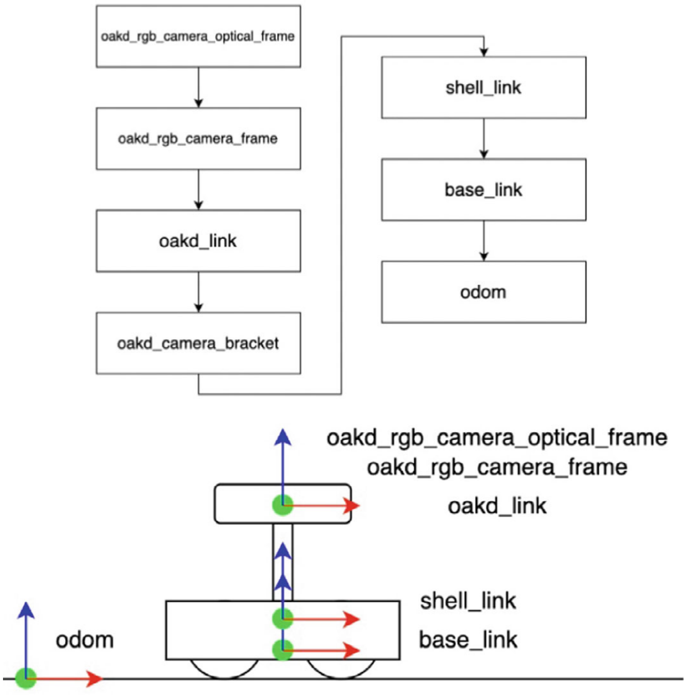


Fig. 2 (Top) The transformation chain of all six coordinate systems. The transformation process begins from the camera's frame to the odometry frame. An arrow represents a transformation. (Bottom) Locations of every coordinate system used in the research

transformation matrix which transfer from frame *b* to frame *a*. Again, since there are six transformations, the final transformation matrix T_6^0 will be the product of the six transformation matrices as illustrated in Eq. 3. Where frame 6 represents "`oakd_rgb_camera_optical_frame`," frame 5 represents "`oakd_rgb_-camera_frame`," and so on and so forth. We can then use the product of the final matrix and the coordinates we get from the camera frame to get H_{odom}, which is the object coordinates in the odometry frame as shown in Eq. 4.

$$T_b^a = \begin{bmatrix} R_{3\times 3}(Q) & \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix}_{3\times 1} \\ O_{1,3} & 1 \end{bmatrix}_{4\times 4} \tag{2}$$

$$T_6^0 = T_1^0 T_2^1 T_3^2 T_4^3 T_5^4 T_6^5 \tag{3}$$

$$H_{odom} = \begin{bmatrix} x_{odom} \\ y_{odom} \\ z_{odom} \\ 1 \end{bmatrix}_{4\times 1} = T_6^0 \begin{bmatrix} x_{camera} \\ y_{camera} \\ z_{camera} \\ 1 \end{bmatrix}_{4\times 1} \tag{4}$$

3.3 Path Planning

After the robot knows where the object actually is, we then need to figure out some path-planning method for the robot to dodge the object. In my case, I use RRT* (Rapidly-exploring Random Trees) as my path-planning method. RRT* is a motion planning algorithm used in robotics and autonomous systems to efficiently find collision-free paths. It builds a tree-like structure by iteratively extending towards randomly selected points while gradually improving the path's optimality. After we get the dodging path planned by RRT*, there is still a problem. The path is usually not smooth enough for the robot to follow properly. Therefore, creating a Bézier curve base on the RRT* path might be a satisfying way to deal with this problem. The Bézier curve interpolates all the points on the path generated by RRT* and therefore smooths out angles on the path.

3.4 Robot Dynamics

The dynamics of the Turtlebot4 can be analogous to that of the unicycle. The main advantage of doing this is that the unicycle model's control inputs are simply linear velocity and angular velocity. That is to say, we only need to deal with two intuitive parameters rather than considering wheel speeds or motor torques directly. According to [9], under the constraints of rolling without slipping, the dynamical model of the unicycle is given by Eqs. 5 and 6, where m is the mass of the robot, I is the inertia of the robot, and the control inputs are the pushing force F and the steering torque N. A purely kinematic model is given by Eq. 5, where the control inputs are the forward velocity v and the angular velocity ω.

$$\begin{aligned} \dot{x} &= v \cos\theta \\ \dot{y} &= v \sin\theta \\ \dot{\theta} &= \omega \end{aligned} \tag{5}$$

$$\begin{aligned} m\dot{v} &= F \\ I\dot{\omega} &= N \end{aligned} \tag{6}$$

3.5 Controller

In accordance with [9], the state of the robot can be expressed in the form of (x, y, θ), where x and y are its coordinates in the odometry frame and θ is its orientation. Assuming that the velocity of the robot's center mass is orthogonal to the wheel axis and the wheels of the robot roll without slipping. Based on the information given, we can get the angular velocity ω and the velocity v for the robot to drive to the destination user assigned by the series of calculations Eqs. 7, 8, 9, 10, and 11, where k_1 and k_2 in Eq. 10 are positive constants. In my case, $k_1 = 1100 \times \left(10^{-4}\right)$ and $k_2 = 5k_1$ when the robot is not dodging the object, $k_1 = 500 \times \left(10^{-4}\right)$ and $k_2 = 20k_1$ when the robot is dodging the object. This process should be done in every driving iteration. In other words, the robot keeps calculating the state difference between its current position and the destination and drives forward for a short distance.

$$\begin{aligned} x &= x_{current} - x_{destination} \\ y &= y_{current} - y_{destination} \\ \theta &= \theta_{current} - \theta_{destination} \end{aligned} \tag{7}$$

$$\begin{aligned} z_1 &= \theta \\ z_2 &= x \cos\theta + y \sin\theta \\ z_3 &= x \sin\theta - y \cos\theta \end{aligned} \tag{8}$$

$$\begin{aligned} x_1 &= z_1 \\ x_2 &= z_2 \\ x_3 &= -2z_3 + z_1 z_2 \end{aligned} \tag{9}$$

$$\begin{aligned} u_1 &= -k_1 x_1 + \frac{k_2 x_3}{x_1^2 + x_2^2} x_2 \\ u_2 &= -k_1 x_2 - \frac{k_2 x_3}{x_1^2 + x_2^2} x_1 \end{aligned} \tag{10}$$

$$\begin{aligned} \omega &= u_1 \\ v &= u_2 + z_3 u_1 \end{aligned} \tag{11}$$

3.6 Implementation

The project can be implemented by a few nodes and topics. "`camera data receiver`" is a node that receives the string coordinates data from the OAK-D

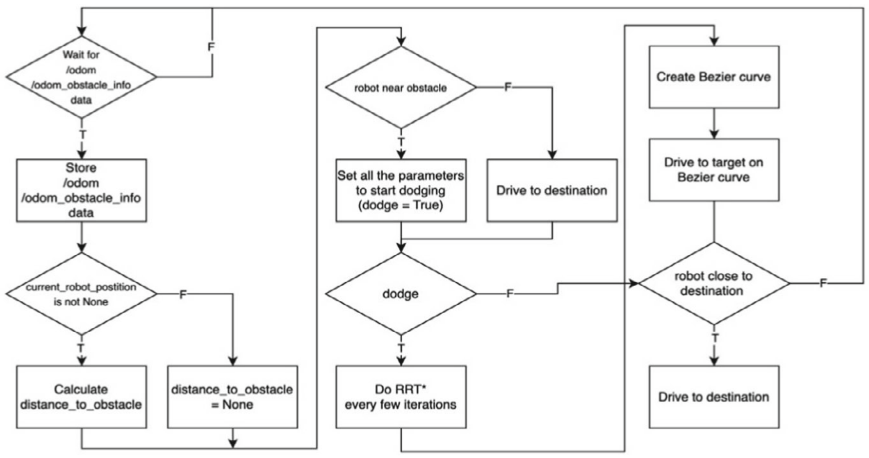

Wait for /odom /odom_obstacle_info data
F
T
Store /odom /odom_obstacle_info data
current_robot_postition is not None
F
T
Calculate distance_to_obstacle
distance_to_obstacle = None
robot near obstacle
F
T
Set all the parameters to start dodging (dodge = True)
Drive to destination
dodge
F
T
Do RRT* every few iterations
Create Bezier curve
Drive to target on Bezier curve
robot close to destination
F
T
Drive to destination

Fig. 3 The flow chart of the dodging process

Pro camera and publishes them to the topic "`/camera_data`." "`camera data reader`" is a node subscribing to the topic "`/camera_data`" and transforming the data into float type. After that, the float data will be published to the topic "`/obstacle_info`." "`transformation`" is a node subscribing to the topic "`/obstacle_info`" and transforming those float coordinates into the coordinates in the odometry frame. At the same time, those data will be published to the topic "`/odom_obstacle_info`." The node "`robot driver`" will subscribe to both "`/odom`" and "`/odom_obstacle_info`" topics at the same time. By doing so, the robot can get its position and the obstacle's position in the odometry frame. Finally, the node will calculate the velocity and the angular velocity of the robot and publish them to the "`/cmd_vel`" topic, through which we can drive the robot. The simulation can subsequently be carried out by adhering to the procedure depicted in Fig. 3 starting from the upper left corner. This is also the working process of the "`robot driver`" node.

4 Results

4.1 Simulation

Before every driving iteration, the robot waits for the new information of `/odom` and `/odom_obstacle_info` to be published. After that, the robot will determine whether the obstacle is close enough to itself. If close, the robot will start dodging the object, or it will keep driving to the final destination the user assigned. The dodging process is comprised of the following parts. First, do the RRT* path planning. Second,

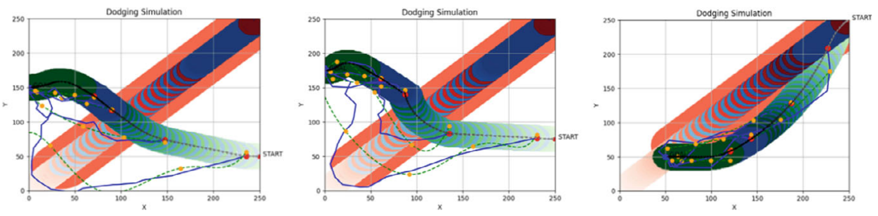


Fig. 4 The robot (shown in green) starts from different points to different destinations. While the obstacle (shown in orange) starts from (0, 0) to (250, 250)

create a Bézier curve based on the RRT* path. Third, drive along the Bezier curve. The dodging process will be done by the robot every several iterations until it is close enough to the final destination. When close to the destination, the robot drives directly to the destination. The robot starts from the right side and the obstacle starts from the lower left corner as shown in Fig. 4. Meanwhile, set the starting point and the final destination of the robot at different points for some experiments. The darkness of the color represents the passing of time. Events that occur earlier are represented by lighter colors.

When the robot and the obstacle both reach the light blue circle in the figure, the robot detects the existence of the obstacle, so the robot starts to dodge the object. At the same time, the robot is on the red dot and starts to create the first RRT* path base on the obstacle map calculated, which is the blue line in the figure. The obstacle map is marked in orange. After that, the robot will create a Bézier curve, which is the green dash line, based on that RRT* path and separates the curve into several sections, the yellow dots are the nodes that separate the curve. The robot then starts driving from the red dot to the first yellow dot in front of it. After a few iterations, the robot does the same process again and again until it is close enough to the final destination. At that point, it will drive directly to the final destination.

4.2 Demonstration Video

In the demonstration video [10], the Turtlebot4 will start driving from the top of the screen to the bottom, as shown in the left image of Fig. 5. The distance I assigned for the robot to proceed is 3 m, through which the robot can reach the destination. After it starts driving, a moving person will walk from the right side to the left side, as illustrated by the right image of Fig. 5, the goal of the robot is to dodge me using the method I designed and then reach the original destination.

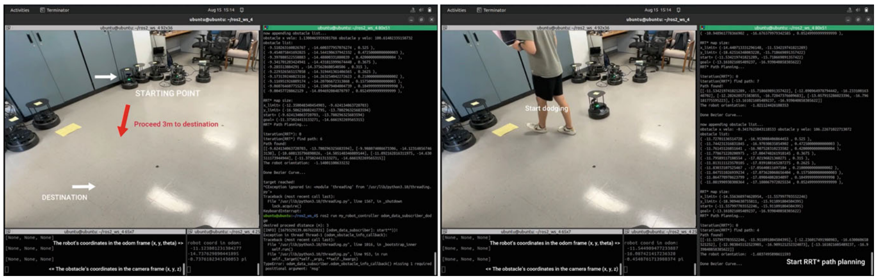

STARTING POINT
Proceed 3m to destination
DESTINATION
The robot's coordinates in the odom frame (x, y, theta) =>
<= The obstacle's coordinates in the camera frame (x, y, z)
Start dodging
Start RRT* path planning

Fig. 5 (Left) The starting point and the destination. (Right) The dodging process starts

5 Conclusion and Future Uses

This research presented a simple method for Turtlebot4 to dodge dynamical obstacles when it is moving. It lays the foundation for affordable, vision-based obstacle avoidance systems. The demonstration video revealed that the methodology developed in this research could perform satisfactorily. The main drawback is that Turtlebot4 sometimes oversteers while dodging, leading to a rough process. Some comparative experiments for the proposed method to compare with LiDARs or event cameras will be conducted to evaluate its performance and benefits.

Contemporary self-driving cars employ LiDARs to acquire information about the obstacles around them. However, LiDARs are much more expensive than cameras. The research elucidated in this report has the potential to be used in self-driving cars, thereby reducing production expenses in the foreseeable future. Besides, if the research illustrated in this project had been employed by unmanned aerial vehicles, in addition to using ultrasonic sensors, drones might have an extra layer of anti-collision protection. In other words, this makes the flights of drones getting safer. More research will be needed to support the feasibility of the potential applications.

Acknowledgements Words cannot express my gratitude to Professor Jorge Cortés and Ph.D. students Neilabh Banzal, Parth Paritosh, and Scott Addams in the MURO Lab at UC San Diego. They provided me with some tips and guidance to help me complete the research. In addition, thanks should also go to the J. Yang and Family Foundation for sponsoring the scholarship and the University System of Taiwan for offering me the opportunity to conduct research at UC San Diego.

References

1. Falanga D, Kleber K, Scaramuzza D (2020) Dynamic obstacle avoidance for quadrotors with event cameras. Sci Robot 5(40):eaaz9712. https://doi.org/10.1126/scirobotics.aaz9712. PMID: 33022598
2. Nguyen T-T-N, Phan T-D, Duong M-T, Nguyen C-T, Ly H-P, Le M-H (2022) Sensor fusion of camera and 2D LiDAR for self-driving automobile in obstacle avoidance scenarios. 2022

International workshop on intelligent systems (IWIS), Ulsan, Republic of Korea, pp 1–7. https://doi.org/10.1109/IWIS56333.2022.9920917
3. Michel P, Chestnutt J, Kuffner J, Kanade T (2005) Vision-guided humanoid footstep planning for dynamic environments. In: 5th IEEE-RAS international conference on humanoid robots, Tsukuba, Japan, pp 13–18. https://doi.org/10.1109/ICHR.2005.1573538
4. Ye N, Wang R, Li N (2021) A novel active object detection network based on historical scenes and movements. Int J Comput Theory Eng 13(3):79–83
5. Kashika PH, Venkatapur RB (2022) Deep learning technique for object detection from panoramic video frames. Int J Comput Theory Eng 14(1):20–26
6. Xie X, Li H, Hu F (2015) The flocs target detection algorithm based on the three frame difference and enhanced method of the Otsu. Int J Comput Theory Eng 7(3):197–200
7. Spatial object tracker on RGB. Luxonis documentation. https://reurl.cc/E1WpzA. Last accessed 28 Oct 2023
8. Siciliano B, Khatib O (2016) Springer handbook of robotics, 2nd edn. Springer Publishing Company (incorporated)
9. DeVon D, Bretl T (2007) Kinematic and dynamic control of a wheeled mobile robot. In: 2007 IEEE/RSJ international conference on intelligent robots and systems, San Diego, CA, USA, pp 4065–4070. https://doi.org/10.1109/IROS.2007.4399599
10. Chu W-T (2023) MURO lab channel. https://www.youtube.com/watch?v=wdz8laYMFX0. Last accessed 28 Oct 2023