# University of California San Diego

## International Summer Research Program

## Robot Dodge Obstacle Based on Camera Feed

August 16, 2023

*Author*

Wei-Teng Chu

# Contents

**1 Research Aim** **3**

**2 Background** **3**
2.1 ROS2 . . . . . 3
2.2 Turtlebot4 . . . . . 3

**3 Development** **5**
3.1 OAK-D Pro Camera . . . . . 5
3.2 Transformation . . . . . 5
3.3 RRT* Path Planning . . . . . 8
3.4 Bézier Curve . . . . . 9
3.5 Controller . . . . . 9
3.6 Simulation . . . . . 10
3.7 Result . . . . . 13

**References** **13**

Figure 1: An example of the Topic structure in ROS2
source

# 1 Research Aim

Assign the robot with a proceed distance which ends up at the destination, and dodge a moving obstacle when it is driving toward it.

# 2 Background

## 2.1 ROS2

ROS2 (Robot Operating System 2) is an open-source framework designed for developing and controlling robots and robotic systems. It provides a flexible architecture that facilitates communication, real-time control, and collaboration between various components of a robotic system. It enables developers to create, simulate, and deploy robotic applications across a wide range of hardware platforms, making it a powerful tool for robotics research and development.

The most frequently-used function of ROS2 in my project is shown in Figure 1. I can break my robot system into several nodes, through which the robot can perform specific tasks. For example, one node collects some data from the Lidar, and the other node process these data. In this case, the one who releases the message is the "Publisher". On the other hand, the one who receives the message is the "Subscriber". However, the message 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 that message from that topic. Of course, there is no limitation that there can only be one Publisher and one Publisher, as the structure shown in Figure 2.

## 2.2 Turtlebot4

I used Turtlebot4 as the robot for my project. Turtlebot4 is a ROS2-based mobile robot intended for education and research. The TurtleBot 4 is capable of mapping the robot's surroundings, navigating autonomously, and running AI models on its camera. Inside the shell sits a Raspberry Pi 4B which runs the TurtleBot 4 software, as shown in Figure 4.

On top of the shell, there is an OAK-D Pro camera and an RPLIDAR A1M8 Lidar. I will only use the camera for my project.

Figure 2: An example of the Topic structure in ROS2
source

Figure 3: Turtlebot4
source

Figure 4: Raspberry Pi 4B in the shell of Turtlebot4
source

Figure 5: OAK-D Pro camera
source

# 3 Development

## 3.1 OAK-D Pro Camera

OAK-D Pro is a stereo camera, 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 pre-trained model, we can track the obstacle in the camera's frame and get the coordinate. As shown in Figure 6 and Figure 7, the camera can track a person in its frame. At the same time, it also provides the coordinates of the object in its own frame.

However, here comes the problem. The robot actually doesn't know where the obstacle actually is, since the coordinates of the robot itself are different from that of the camera. Therefore, we need some transformations here.

## 3.2 Transformation

As you can see in Figure 8, 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

Figure 6: OAK-D Pro camera tracking person

Figure 7: OAK-D Pro camera tracking person

Figure 8: An example of transformation
source

Figure 9: Transformation chain of all six transformations

is p'.

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 Figure 9, the transformation starts from "oakd_rgb_camera_optical_frame" all the way through "odom."

All the frames I mentioned and their locations are shown in Figure 10. 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, I do not show all of them.)

As I mentioned before, we can get the transformation information from the topic. Actually, what we are

Figure 10: Locations of every coordinates system
source

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 Equation 1, where $q_0$, $q_1$, $q_2$, and $q_3$ are the quaternions.

$$R(Q) = \begin{bmatrix} 2(q_0^2 + q_1^2) - 1 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & 2(q_0^2 + q_2^2) - 1 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & 2(q_0^2 + q_3^2) - 1 \end{bmatrix} \tag{1}$$

After that, we put the rotational matrix into the upper left corner of a 4x4 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 Equation 2. By doing so, we can create a transformation matrix between two frames. Again, since there are six transformations, the final transformation matrix will be the product of the six transformation matrices.

We can then use the product of the final matrix and the coordinates we get from the camera frame to get $H_{transform}$, which is the object coordinates in the odometry frame as shown in Equation 2.

$$(H_{transform})_{4\times1} = \prod_{k=1}^{6} \begin{bmatrix} R_{3\times3}(Q) & \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix}_{3\times1} \\ O_{1,3} & 1 \end{bmatrix}_{k,\ 4\times4} \begin{bmatrix} x_{camera} \\ y_{camera} \\ z_{camera} \\ 1 \end{bmatrix}_{4\times1} \tag{2}$$

## 3.3 RRT* 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 actually dodge the object. In my case, I use RRT* (Rapidly-exploring Random Trees Star) as my path-planning method, which is shown in Figure 11. 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.

Figure 11: RRT* path planning
source

### 3.4 Bézier Curve

I decided to create a Bézier curve base on the RRT* path. In this way, the curve does a kind of interpolation to the RRT* path, which therefore smooths out the sharp angle on it. An example of Bézier curve is shown in Figure 12.

### 3.5 Controller

The control of the turtlebot4 can be analogous to that of the unicycle. According to the paper [1], the state of the robot can be expressed in the form of $(x, y, \theta)$, where $x$ and $y$ are its coordinates in the odometry frame and $\theta$ 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.

Figure 12: An example of Bézier curve
source

Based on the information given, we can get the angular velocity $\omega$ and the velocity $v$ for the robot to drive to the destination user assigned by a series of calculations starting from Equation 3 all the way through Equation 7, where $k_1$ and $k_2$ in Equation 6 are positive constants. In my case, $k_1 = 1100 \cdot (10^{-4})$ and $k_2 = 5k_1$ when the robot is not dodging the object, $k_1 = 500 \cdot (10^{-4})$ 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.

$$\begin{aligned} x &= x_{current} - x_{destination} \\ y &= y_{current} - y_{destination} \\ \theta &= \theta_{current} - \theta_{destination} \end{aligned} \tag{3}$$

$$\begin{aligned} z_1 &= \theta \\ z_2 &= x\cos\theta + y\sin\theta \\ z_3 &= x\sin\theta - y\cos\theta \end{aligned} \tag{4}$$

$$\begin{aligned} x_1 &= z_1 \\ x_2 &= z_2 \\ x_3 &= -2z_3 + z_1 z_2 \end{aligned} \tag{5}$$

$$\begin{aligned} u_1 &= -k_1 x_1 + \frac{k_2 x_3}{x_1^2 + x_2^2} x_2 \\ u_2 &= -k_1 x_2 - \frac{k_2 x_3}{x_1^2 + x_2^2} x_1 \end{aligned} \tag{6}$$

$$\begin{aligned} \omega &= u_1 \\ v &= u_2 + z_3 u_1 \end{aligned} \tag{7}$$

### 3.6 Simulation

I do the simulation following the process shown in Figure 13 starting from the upper left corner.

Before every 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, create a Bezier curve based on the RRT* path. Third, drive along the Bezier curve. The dodging process will keep 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 Figure 14, Figure 15 and Figure 16. Meanwhile, I set the final destination of the robot at different point in Figure 14, Figure 15 and Figure 16. The darkness of the color represents the passing of time. The lighter the color, the earlier the event occurs.

Figure 13: The flow chart of the dodging process

Figure 14: The simulation of the dodging process

Figure 15: The simulation of the dodging process

Figure 16: The simulation of the dodging process

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.

### 3.7 Result

In the demo video, the turtlebot will start driving from the top of the screen to the bottom. The distance I assigned for the robot to proceed is 3 meters, 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, the goal of the robot is to dodge me using the method I designed and then reach the original destination.

## References

[1] D. DeVon and T. Bretl, "Kinematic and dynamic control of a wheeled mobile robot," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4065–4070, 2007.