ECE 470 / ME 445 Final Project Report

An Autonomous Fire-Sensing Robot on V-REP

Manan Mehta (mananm2)

Mehul Kaushik (mehul2)

Team FireBot

https://github.com/mananm2/FireBot_ece470

https://www.youtube.com/watch?v=iA9K4rRUnZg

Submitted on December 15, 2019

Abstract

The project aims at developing an autonomous fire-sensing robot on V-REP which can sense fire in a given layout and extinguish it. Through this, we aim to reduce the time lag between fire sensing and the arrival of firemen, thereby containing most fires that start out small. This has potential applications in several indoor environments like laboratories, warehouses, factories, hospitals and even homes. We developed a simulation prototype for this project by making a robot autonomously follow the left wall of an indoor layout while simultaneously sensing for fire. As soon as our robot senses the fire, it stops and moves its arm to the base of the fire to extinguish it. The robot follows a control logic in wall following and maintains an optimal distance between 0.9 and 1.1 meters from the wall at all times. Further, the sensor space – the total area covered by the front and right sensors – covers 100% of the indoor map, thus leaving no blind spots in fire detection. In the future, we intend to add modules for intelligent motion planning in any general layout for the robot, thus improving the efficiency of robot motion. Through the project, we applied concepts like Robot Control, Sensing and Inverse Kinematics to a real-world problem – thus reinforcing our in-class knowledge.

1. Introduction

The goal of our project was to create an intelligent robot that immediately act against fire hazards on site and hence reduce response times for fires. Not only do we wish to curb response times but also have an onsite surveillance system that integrates a robot to better survey the lay the land before the firefighters arrive hence allowing firefighters to better prepare against crisis.

According to FEMS, the response times are maintained at about 5 minutes and 20 seconds [1]. We believe our own solution can further reduce this to practically zero as the robot already present on site can immediately begin emergency procedures that include escorting civilians and even moving hazardous debris along with extinguishing fires. We focused mainly on the later most given our limited resources and skills at designing such a high-level robot. We are also well aware of many already existing robots out there in the field such as THOR, Thermite robot, TAF 20 and Fire Ox [2]. All these robots are heavyweights that can take on major fires that can be hard to control within an hour but our solution is slightly different in that this robot we are planning may stow at any possible location instead of far off at the fire station. These smaller robots will have better maneuverability in closed spaces and can take on smaller fires and collect data before fire fighters are even called to the scene.

We thought of many design considerations but the best suited one we believed would include a robot arm and one that implements computer vision using infrared and ultrasound like sensors since fire is easily detected when judged using color temperature readings in computer vision [3]. We attempted to make best use of infrared detection as possible using our own knowledge of blob detection as possible though it is important to note that even our own project doesn't really manage to differentiate hazardous fire from non-hazardous, which is a possible area for future research and has great scope for improvement in terms of enhancing our own design.

As a team, we started this project with an initial concept sketch which eventually led us to successfully build FireBot, as shown in Figure 1.1.

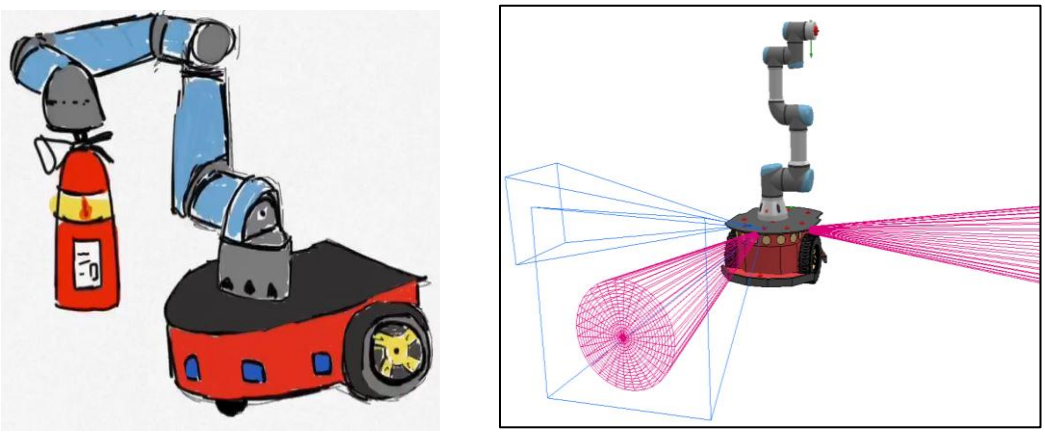


Figure 1.1: Initial concept sketch of FireBot and the final robot built on V-REP

2. Method

2.1 Block Diagram

Figure 2.1 shows the block diagram prepared for the project. The robot, named 'FireBot', is divided into two separate modules. The Pioneer P3dx two-wheeled mobile robot on V-REP was used to move around in a layout by controlling the velocities of the left and right wheels. The UR3 robot arm was mounted on top of the mobile robot to carry a fire hose or an extinguisher. The Pioneer P3dx was mounted with four sensors in total – two proximity sensors for wall following and two vision sensors for fire sensing.

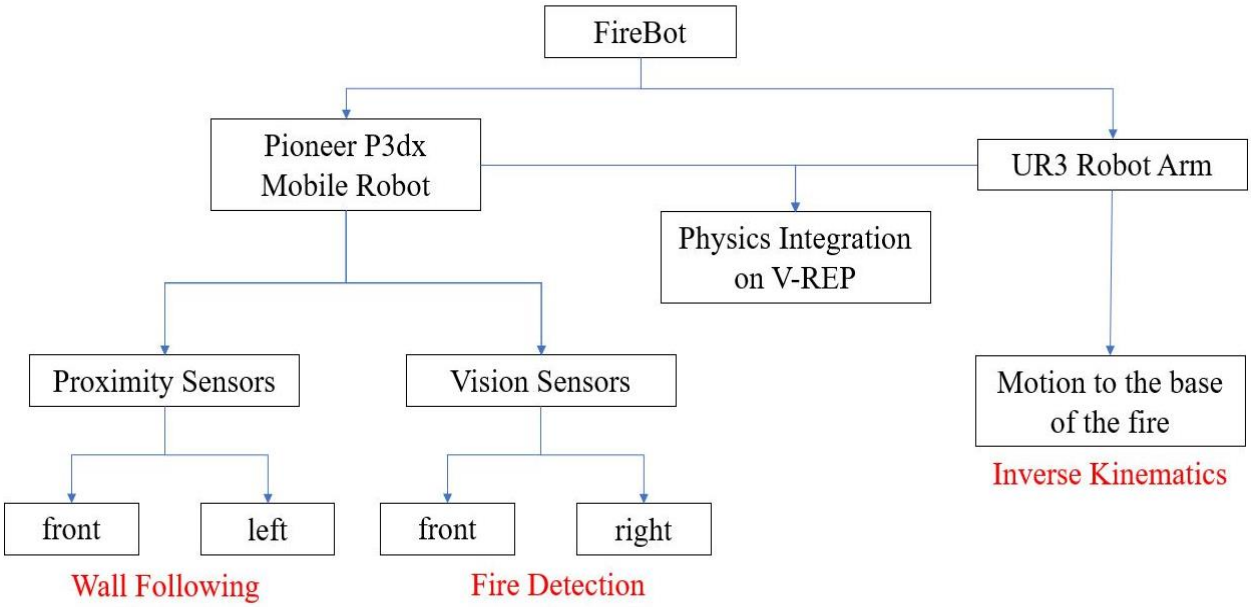


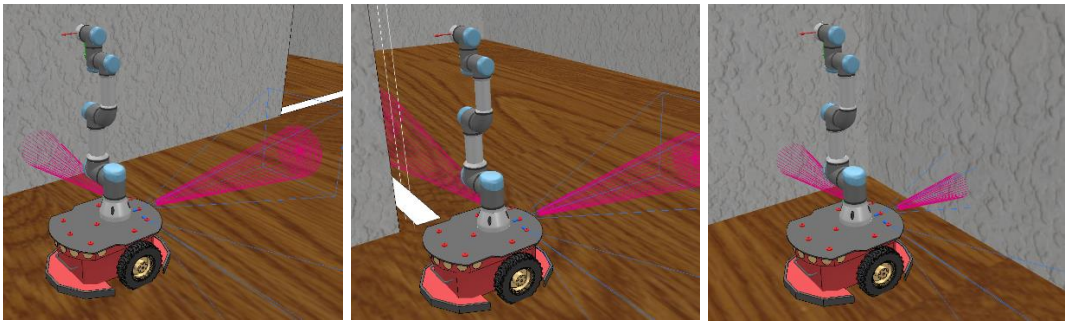
Figure 2.1: Block Diagram for FireBot

2.2 The Pioneer P3dx Mobile Robot and Wall Following

The idea behind wall following is to make the robot traverse the entire indoor layout at least once. The designed indoor layout for our project (a test layout) is shown in Figure 3.1. For a practical case, this can be thought of as a sequential step in fire safety. If fire is detected anywhere in the layout using overhead sensors or smoke detectors, the robot can start its motion and search the layout for the fire location.

The Pioneer P3dx two-wheeled robot is available in V-REP mobile robots. The robot takes in two separate velocities to the left and right wheels. We use a simple case-based control logic to govern how the robot follows the left wall. Two proximity sensors are mounted on the robot which sense the wall to its left and front. We consider that the proximity sensors read TRUE if they sense a wall and FALSE otherwise.

For a simple understanding of the wall following logic, refer to Figure 2.2 along with the pseudocode given in Figure 2.3.



(a) (b) (c)

Figure 2.2: Wall following decision logic implemented using two sensors: (a) Keep going straight (b) Turn left or around the wall and (c) Turn right

```
start
left wheel velocity= right wheel velocity
while fire_not_detected
        if left_sensor = TRUE and front_sensor = FALSE (Fig 2.2 a)
                keep going straight
                left wheel velocity= right wheel velocity
        else if  left_sensor = FALSE and front_sensor = FALSE (Fig 2.2 b)
                wall has ended
                turn left
                left wheel velocity < right wheel velocity
                end turn when left sensor becomes normal to the left wall
        else if  left_sensor = TRUE and front_sensor = TRUE (Fig 2.2 c)
                wall in front
                turn right
                left wheel velocity > right wheel velocity
                end turn when left sensor becomes normal to the left wall
        if fire_detected
                stop robot
                left wheel velocity = right wheel velocity = 0
```

Figure 2.3: Pseudocode for wall following logic

The pseudocode in Figure 2.3 explains the wall following qualitatively, however, the tuning was done quantitatively. The initial wheel velocity governed the difference in left and right wheel velocities for left and right turns. This fine tuning is explained in Section 3.1.

Next, we explain the vision sensors and the image processing used to detect fire.

2.3 Vision Sensor and Fire detection

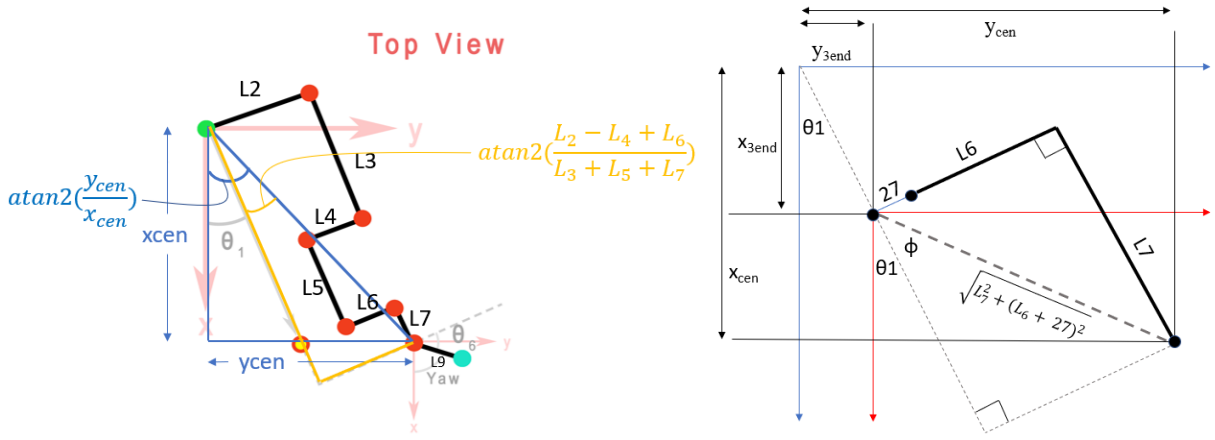
Like the two proximity sensors used for wall following, we use two vision sensors for fire detection – one on the front of the robot and one on the right. The challenge of detecting fire is to isolate it in contrast to everything else visible to the robot in the background. One solution for this was to reduced the range of the sensor to ensure room objects are not detected. However, in this configuration, the entire indoor layout would not be covered and the two sensors would not map the entire floor area exhaustively.

Figure 2.4: Vision Sensor filter for fire detection

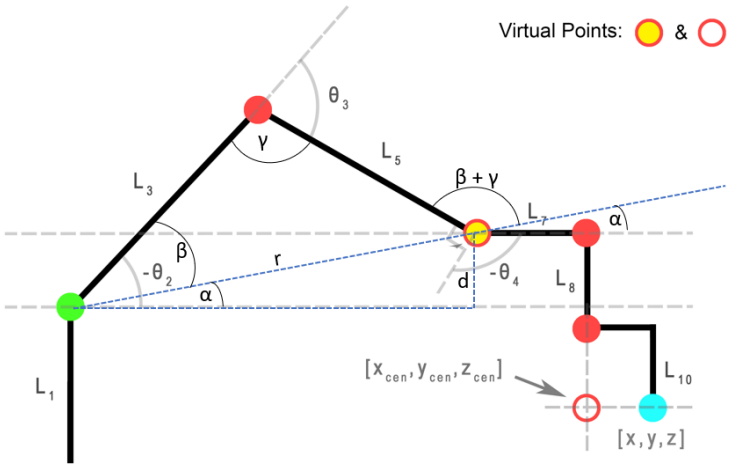
To solve this, we used the image processing available in the Vision Sensor Module in V-REP. This filter helps us extract only the red pixels from an image as seen in Figure 2.4. This ensures that the robot only stops on detecting fire and no other object in the room. The details of extracting the red pixels and applying this filter are discussed in Section 3.2.

2.4 The UR3 Robot Arm and Inverse Kinematics

For inverse kinematics, we used the equations derived in Lab 5 of the course. For completeness, we show the UR3 geometry here in Figure 2.5 (a) through (e) and write the equations for the joint angles. For a detailed derivation of the same, refer to the Lab 5 report.



(a) (b)



(c)

Figure 2.4: (a) θ1 and θ6 calculation (b) Calculation of (x_{3end}, y_{3end}, z_{3end}) and (c) Side-View of the robot arm for calculation of θ2, θ3 and θ4

The following six joint angles were obtained for the UR3:

$$\boldsymbol{\theta_1 = atan2\left(\frac{y_{cen}}{x_{cen}}\right) - atan2\left(\frac{L_2 - L_4 + L_6}{L_3 + L_5 + L_7}\right)}$$

$$\boldsymbol{\theta_2 = -(\alpha + \beta)}$$

$$\boldsymbol{\theta_3 = \pi - \gamma}$$

$$\boldsymbol{\theta_4 = 2\pi - \left(\frac{\pi}{2} + \alpha + \beta + \gamma\right) = \frac{3\pi}{2} - \alpha - \beta - \gamma}$$

$$\boldsymbol{\theta_5 = -\frac{\pi}{2}}$$

$$\boldsymbol{\theta_6 = \theta_1 + \frac{\pi}{2} - yaw}$$

where

$$x_{3end} = x_{cen} - \sqrt{L_7^2 + (L_6 + 27)^2}\cos(\theta_1 + \varphi)$$

$$y_{3end} = y_{cen} - \sqrt{L_7^2 + (L_6 + 27)^2}\sin(\theta_1 + \varphi)$$

$$z_{3end} = z_{cen} + L_8 + L_{10}$$

$$\varphi = atan2\left(\frac{27 + L_6}{L_7}\right)$$

$$d = z_{3end} - L_1$$

$$r = \sqrt{x_{3end}^2 + y_{3end}^2 + d^2}$$

$$\alpha = \sin^{-1}\left(\frac{d}{r}\right)$$

$$\beta = \cos^{-1}\left(\frac{L_3^2 + r^2 - L_5^2}{2L_3 r}\right)$$

$$\gamma = \cos^{-1}\left(\frac{L_3^2 + L_5^2 - r^2}{2L_3 L_5}\right)$$

We changed certain things from Lab5, the most important of them being the frame of reference. The world coordinates were set at the base of the UR3 in V-REP and thus, we had to take this into account while writing the Python code. Further, depending on where the fire is detected, the UR3 arm had to move to the base of the fire. This was done by giving the vision sensor reading to the UR3 and accordingly moving it in front or to the right.

Finally, the modules explained in Sections 2.1 through 2.4 were integrated by altering the physics of each module for them to function in tandem. These changes were important for the model to run smoothly and are discussed in Section 3.3.

3. V-REP Setup and Parameter Tuning

The Figure 3.1 shows the setup created on V-REP for the simulation. An indoor layout was modeled using cuboids as walls. A fire was simulated at a random location inside the layout. The robot starts in the position shown and moves around the layout to detect the fire.

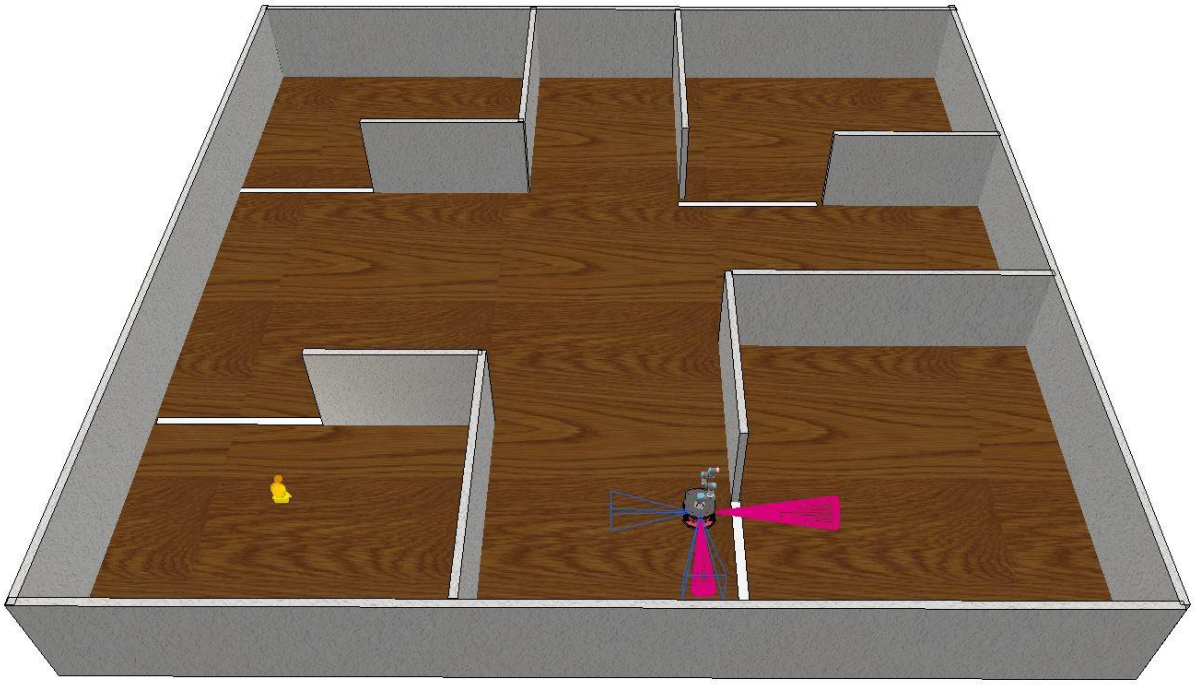


Figure 3.1: The indoor layout modeled on V-REP

The system was simulated using a Python API script coupled with V-REP. The simulation is connected to the code by specifying the ClientID in the code. If the simulation fails to connect, the script returns an error and quits as shown in Code Snippet 3.1. Each component of the model was accessed by obtaining their respective object handles, as shown in Code Snippet 3.2. The same was repeated to get the handles of each joint of the UR3 robot arm.

```
clientID=vrep.simxStart('127.0.0.1',19999,True,True,5000,5)
if clientID!=-1:
        print ('Connected to remote API server')
else:
        print ('Connection not successful')
        sys.exit('Could not connect')
```

Code Snippet 3.1

```
errorCode, Left_Motor =
vrep.simxGetObjectHandle(clientID,'Pioneer_p3dx_leftMotor',vrep.simx_opmode_blocking)

errorCode, Right_Motor =
vrep.simxGetObjectHandle(clientID,'Pioneer_p3dx_rightMotor',vrep.simx_opmode_blocking)

errorCode, leftsensor =
vrep.simxGetObjectHandle(clientID,'Proximity_sensor_left',vrep.simx_opmode_blocking)

errorCode, frontsensor =
vrep.simxGetObjectHandle(clientID,'Proximity_sensor_front',vrep.simx_opmode_blocking)

errorCode, frontcam = vrep.simxGetObjectHandle(clientID,'frontcam',vrep.simx_opmode_blocking)

errorCode, rightcam = vrep.simxGetObjectHandle(clientID,'rightcam',vrep.simx_opmode_blocking)
```

Code Snippet 3.2

Each of these object handles were used in appropriate locations to achieve specific tasks, like setting wheel velocities, reading proximity and vision sensor data and the like.

3.1 Tuning Wheel Velocities

After several trail and errors, the robot's initial wheel velocities were decided to be 2.5 m/s for each wheel to move in a straight line. Table 3.1 summarizes the velocities of left and right wheel for both turns.

Motion / Case	Left Wheel Velocity	Right Wheel Velocity
Straight Line	Speed (=2.5m/s)	Speed
Left Turn	-0.1 x Speed	0.5 x Speed
Right Turn	Speed	0.0 x Speed
Distance to Wall > 1.1 m	0.6 x Speed	Speed

Table 3.1: Turning velocities for each wheel and turn

We end the turn(s) when the left sensor of the robot is normal to the left wall after the start of the turn. However, in some cases, there is a time lag between the end of the turn command and the actual switching of velocities back to equal. This may lead the robot to turn more, thus moving away from the wall. To account for this, we have the fourth row - when the distance exceeds 1.1 meters, the robot moves slightly to the left until the distance is within (0.9,1.1) meters again. This is a simple implementation of P-control for distance.

It must be noted that the parameters obtained in Table 3.1 are for a straight-line speed of 2.5 m/s. If the straight-line speed is changed, the parameters have to be tuned again.

3.2 Setting up the Vision Sensor

The vision sensor had to be isolated for red pixels only, to ensure fire detection. The end result of this is shown in Figure 2.4. For this, we first change the render mode of the vision sensor in object properties from OpenGL (default) to OpenGL - auxiliary channels only. This is then readable using the simxReadVisionSensor remote API in python. The function returns an array of 15 auxiliary values in the form shown below, each of which ranges from 0 to 1:

{min (intensity, red, green, blue, depth), max (intensity, red, green, blue, depth), average (intensity, red, green, blue, depth)}

Thus, for only high intensity red light (fire) detection, the seventh value (maximum value of the red pixel) becomes 1.0. This is when we say that fire has been detected.

3.3 Physics Integration

Since FireBot contains two separate robots integrated together (Pioneer P3dx and UR3), it was essential to alter the physics for the assembly to work in tandem:

- We changed the size of the entire arena to the size of the UR3, as the Inverse Kinematics only works in the default size of the UR3. Scaling doesn't help either, as the coordinate frames do not get scaled.
- We had to change the masses and moments of inertia of each robot link. As the model was scaled, the Pioneer P3dx was unable to balance the arm on top. We changed scaled the masses of the links down and switched off the force interaction between the base of the UR3 and the top of the Pioneer P3dx.
- We altered the joint velocities of joints 2 and 4 in the UR3. This is because the default velocities were too high and were causing the arm to move instantaneously, which is impossible in real scenarios.
- We made the arm non-detectable under the proximity sensors. Before we did this, the proximity sensors always detected the arm and the robot kept spinning around its own axis!
- There is no fire extinguisher model on V-REP and thus we couldn't extinguish the fire. However, moving the arm at the base of the fire was sufficient to show the working of our design.

4. Data and Results

On putting everything together, the final model ran smoothly and was able to detect a fire anywhere in the given layout. Figure 4.1 shows a successful fire detection with the arm pointing to the base of the fire. For a full wall following detection, watch the video linked on the title page.

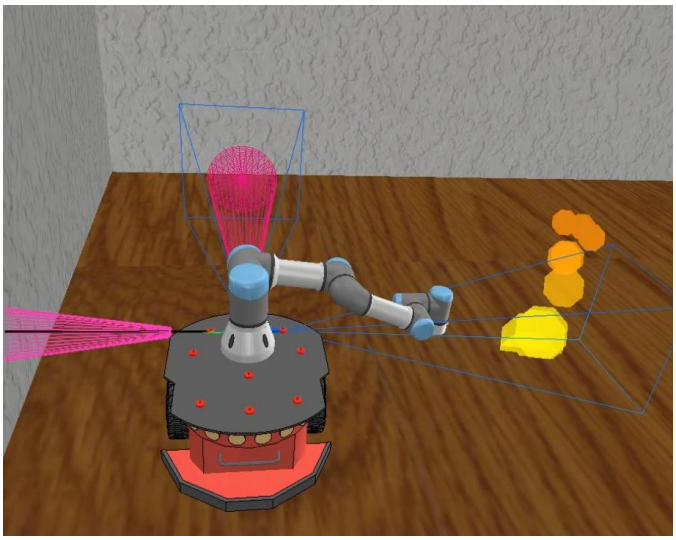


Figure 4.1: Snapshot after a successful run (Qualitative Example of Success)

We have two main quantitative aspects to analyze – the success of the control logic to maintain the robot on its path and the errors in the inverse kinematics.

4.1 Quantitative Analysis of Wall Following

Figure 4.2 shows a time-plot of the distance of the robot from the left wall. The reading was taken using the graph functionality in V-REP and exported to a CSV file where it was plotted.

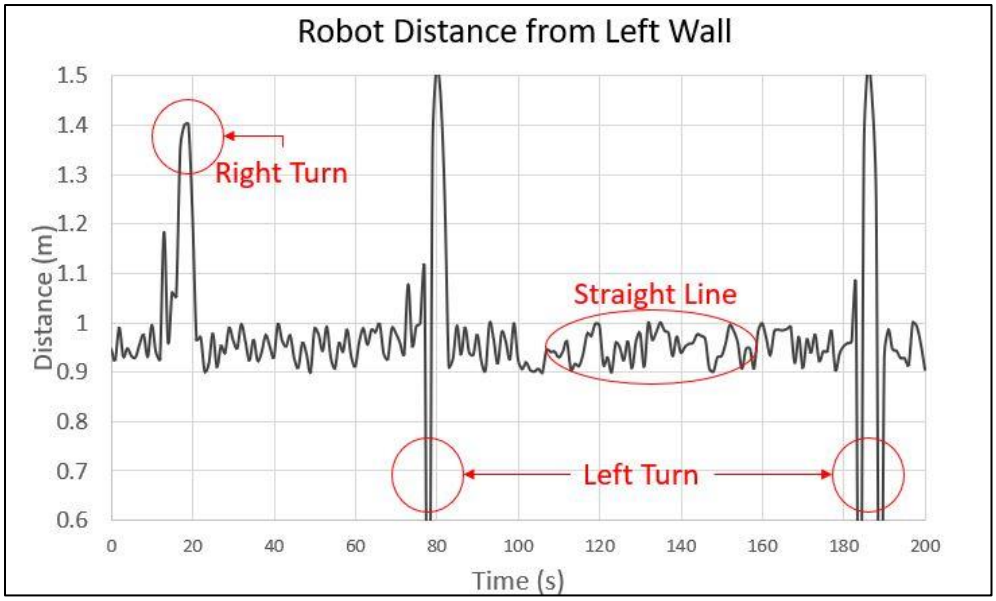


Figure 4.2: Distance of the robot from the left wall vs. time

In Figure 4.2, The Y-axis shows the distance measured by the left sensor from the left wall and the X-axis is time in seconds. As we can see, the robot is maintained at a distance between 0.9 and 1.2 meters from the left wall at all times which means our wall following logic has no error. The spikes in the plot arise when the robot is turning and can be explained as follows:

- Right Turn: The distance from the wall increases first and then decreases as the robot is turning right. This can be visualized in Figure 2.2 (c) and occurs at t = 20 sec in the plot.
- Left Turn: At the onset of left turn, the robot does not read the wall (as there is no left wall!) and thus the reading goes to zero. As we start turning left, the sensor again comes in contact with the wall and turns either 90˚ or 180˚. This can be visualized in Figure 2.2 (b).

4.2 Quantitative Analysis of Inverse Kinematics

The next source of error is the inverse kinematics of the UR3 arm. We tested our inverse kinematic results by moving the robot to the positions we feed it using our new python function that we designed and measure by how much is it off using Euclidean distances. The results are summarized in Table 4.1.

Input Value (mm,degrees)	Calculated (degrees)	Measured value (mm)	Error (mm)
(200,200,200,yaw=90)	(-12.1, -98.1, 112.4, -14.3, -90, 76.3)	(190, 200, 205)	11.125
(200,50,30,yaw=0)	(-39.2, -63.4, 132.6, -69.2, -90, 50.8)	(190, 60, 35)	14.823

Table 4.1: Inverse Kinematics measurements

These results are surprisingly close to what we obtained in Lab 5. We predict that these consistent offsets between the V-REP simulation and Lab 5 may be due some constant error in values. This may arise due to round off somewhere or in the assumptions we made while solving the inverse kinematic problem. Nonetheless, considering the scale of our problem (meters), an error of ~10 mm in the robot arm position is not detrimental to the functioning of the robot.

4.3 System Performance – Success and Failure

We used several failed cases to build a successful system. For instance, the system did not perform well when turning velocities were not tuned. Although the system started performing better when we started tuning the velocities, getting an optimum was required. Since the system is stochastic to a certain extent, we tuned the velocities till the point that the possibility of failure was close to 0. Similarly, we scaled the UR3 arm to accommodate for the physical interaction between the 2 robots (jerk motion due to the UR3 movement on the Pioneer P3dx). This, again, was a part of optimization. The system may fail if a faster velocity is provided as an input as the tuning fails. It may also fail for a bigger layout where the length of the vision sensor may not accommodate the entirety of the layout area. This can be tackled by building an autonomous robot with a sophisticated computer vision algorithm for motion planning.

5. Conclusion

To sum up, we started from zero knowledge of V-REP (and robotics) and worked our way through the semester to build a successful simulation of a real system from scratch. From the data, we see that the robot is robust under most circumstances and can be very useful in applications discussed in Section 1.

In the process of building this project, we reinforced the concepts we learned in class by applying them to a real-world scenario. We started with the idea of using particle filters to estimate robot position in the layout. However, being a two-member team, we decided to go with a wall-following approach by using proximity sensors. Nonetheless, we applied concepts from sensing and estimation to work our way around the problem. The wall following was the most difficult part of the project but it also helped us learn the most – both about robot control and about V-REP.

This was followed by implementing Inverse Kinematics that we learned in class and in Lab 5. Although we borrowed most concepts from Lab 5, the challenge here was to integrate the UR3 arm with the moving robot. This is where conflicting physics started to cause trouble. The physics of the simulation automatically changed in the beginning on sensing fire, leading to absurd results as discussed in Section 4. However, meticulously analyzing the errors helped us solve those issues to finally obtain a fully working robust model.

We can think of several modifications and add-ons for the project if it was to be continued further or done again. We list down a few as follows:

- Use a robust PID control logic for wall-following. This would eliminate the requirement to tune the turning wheel velocities every time the base speed is changed. A PID controller would be able to do the tuning ‘on the fly.’
- Make the robot fully autonomous by implementing multiple vision sensors and using Computer Vision. This would be more efficient as the robot would not be ‘constrained’ by wall-following and would move ‘freely’ in the indoor layout searching for fire. This can further improve response times.
- From a practical standpoint, a fire-sensitive indoor environment is bound to have overhead or ceiling fire detectors. We can couple them with the robot to narrow down the amount of fire ‘searching’ the robot has to undergo. For instance, the ceiling fire detectors can detect a fire originating near/around Room 3 (say) and parse that to the robot. The robot can then skip Rooms 1 and 2 and directly investigate Room 3. This, again, can intelligently improve response time.

Overall, the project (vastly) augmented our in-class and in-lab learnings.

6. References

[1] https://fems.dc.gov/page/fire-response-time - Fire Response Time, Washington DC

[2] https://safetymanagement.eku.edu/blog/the-use-of-robotics-in-firefighting/ - Robots in Firefighting, EKU Online

[3] Zaman, et al.: Fire Detection Using Computer Vision (https://ieeexplore.ieee.org/document/8623842)

[4] Madhevan, et al.: Modelling, Simulation and Mechatronics Design of a Wireless Automatic Fire Fighting Surveillance, Def. SCI. J., Vol. 67, No. 5, September 2017

[5] https://github.com/nettercm/trinity – Trinity Fire Fighting Contest Github

[6] http://forum.coppeliarobotics.com/viewtopic.php?t=2579 – Wall Following Robot Forum, Coppelia Robotics

[7] https://github.com/ssscassio/ros-wall-follower-2-wheeled-robot/tree/master/report – Cassio Santos, Control Algorithms for a wall following 2 Wheel Robot