TP2 ROS
Mission Coordination
M2 SAAS

Prepared by:

Student Name	Student Number
Mohamed Mustafa	20246584
Shehab GOMMA	20246370

Repository link: https://github.com/Mohamed-Tarek-18/Go_To_Flag_Robot.git

Under The Supervision Of:

Prof. Sofiane Ahmed-Ali

PhD student: Boris KIEMA

1 Proposed Solution

Our strategy depends on using PID controlling technique for guiding the robots at each corresponding flag. The first one calculates the error as the Euclidean distance between the robot position and a flag coordinate. And the second adjusts the hading angle to directly face the corresponding flag.

For collision between robots a timing strategy is used which is to start the robots' motion each at a different time.

In avoiding colliding with the obstacles, reading from the sonar sensor is utilized. If the sonar reads an object it turns for a specified time and direction, then it let the PID controllers readjust the heading of the robots after avoiding the obstacles.

2 Limits of the least robust solutions

the Obstacle avoidance solution is limited with the size of the obstacles. It works well with relatively small obstacles (nearly the size of the robot.) for bigger obstacles the turn distance needs to be bigger. The sonar is also limited with a range and bigger obstacles will need to be detected from a bigger distance ahead.

3 Improvements

The improvements That Should Be added is making the Obstacle Avoidance Controller Based despite of Timing Strategy and this is done by considering a relation between the obstacle size and the deviation angle also from the optimization point of view this control based must select the optimal deviation angle to achieve an optimal path to minimize the cost. This from my point of view can be done by attaching the sonar sensor on a dynamical moving hinge make it scanning at different angles until it does not find the obstacle at which angle and go with this angle until avoiding it.

4 Explanation of each solution

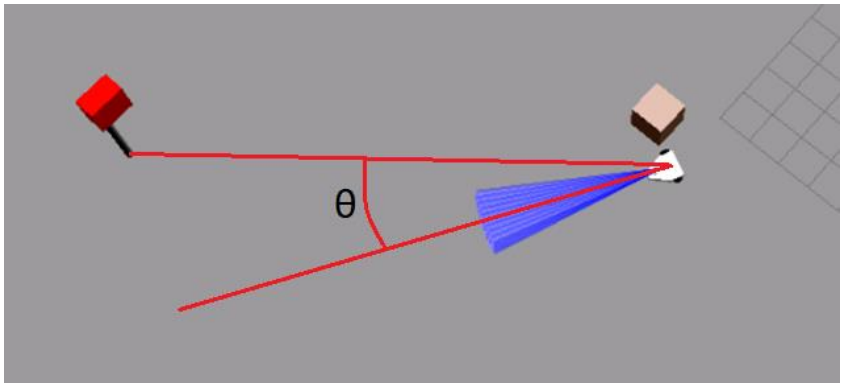
4.1 Navigation to the flag

To implement the proposed controllers the service to get the distance from the flag has been edited to return the coordinates of the flag the edits have been to the files: distance_to_flag_plugin.cpp (found in the plugins directory) and DistanceToFlag.srv

Using the flag coordinates and the robots coordinates the deviation angle between the robot heading and the flag direction using arctan2 function this deviation angle is feedbacked with PID Calculations to the angular velocity command to adjust the heading of the robot.

$$angle = \tan^{-1}\frac{y_{flag} - y_{robot}}{x_{flag} - x_{robot}}$$

$$\theta = angle - robot\ heading$$



The flag distance is feedbacked as the velocity command with PID controller Calculations to the Linear velocity command. When the distance is large the velocity will be large and gets closer to the flag the robot slows down and Stop on the flag. Code is included in the appendix

$$flag\ distance = \sqrt{\left(x_{flag} - x_{robot}\right)^2 + \left(y_{flag} - y_{robot}\right)^2}$$

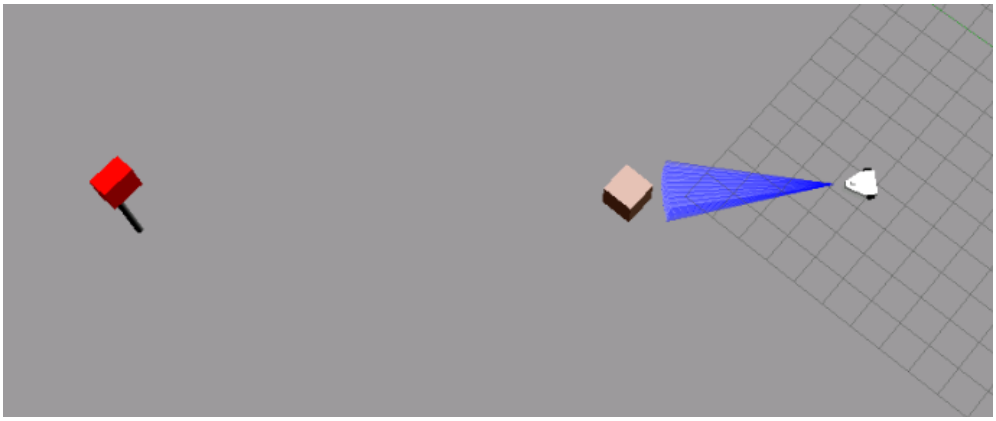
The PID controllers are as follows

$$v_{angle} = K_p * angle\_err + ki * (angle\_err\_old + angle\_err) + kd * \frac{angle\_err - angle\_err\_old}{dt}$$

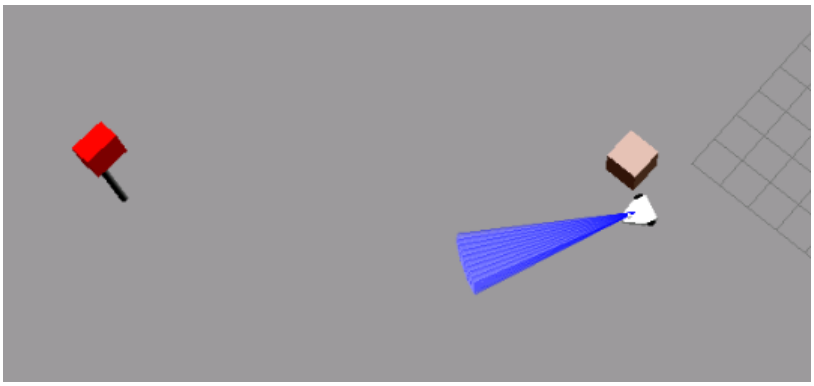
$$v_{linear} = kp * err + ki * (err\_old + err) + kd * \frac{err - err\_old}{dt}$$

4.2 Avoid obstacles

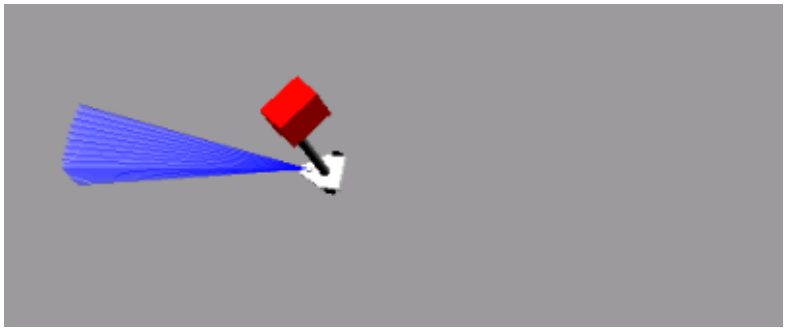
In avoiding obstacles, the reading of the sonar is obtained and used to detect an obstacle.



Once an obstacle is detected a predefined angular and Linear Velocity Commands will drive the robot Away from the obstacle



Then from this point The PID controllers will work again and guide the robot to the flag



5 Appendix

5.1 Agent.py code

5.1.1 GitDistanceToFlag modified part

```
    def getDistanceToFlag(self):
        """Get the distance separating the agent from a flag.
        The service 'distanceToFlag' is called for this purpose.
        The current position of the robot and its id should be specified.
        The id of the robot corresponds to the id of the flag it should reach

        Returns:
            float: the distance separating the robot from the flag
        """
        rospy.wait_for_service('/distanceToFlag')
        try:
            service = rospy.ServiceProxy('/distanceToFlag', DistanceToFlag)
            pose = Pose2D()
            pose.x = self.x
            pose.y = self.y
            # int(robot_name[-1]) corresponds to the id of the robot.
            It is also the id of the related flag
            result = service(pose, int(self.robot_name[-1]))
            return result
        except rospy.ServiceException as e:
            print("Service call failed: %s" % e)

    def PID_control(self, kp, ki, kd):
        #plaplapla
        self = []
```

5.1.2 the strategy code

```
def run_demo():
    """Main loop"""
    robot_name = rospy.get_param("~robot_name")
    robot = Robot(robot_name)
    print(f"Robot : {robot_name} is starting..")




    # Timing between robots
    rospy.sleep(3 * int(robot_name[-1]))
```

```
    # Strategy:

    # PID coefficients
    kp = 1.2
    ki = 0.09
    kd = 0.6

    # Initialize algorithm parameters
    vt = 0
    angle = 0
    d_des = 0
    err_old = 0
    angle_err_old = 0
    dt = 0.05       # delay for derivative in PID calculation line

    # Initialize the robot speed
    robot.set_speed_angle(vt, angle)

    # get the flag ditance and pose:   We edited the DistanceToFlag
    service and code to obtain the pose
    flag1 = [robot.getDistanceToFlag().flag_x, robot.getDistanceToFlag().flag_y]
    print(f"{robot_name} Flag coordinates = ", flag1[0] , " ", flag1[1])

    # start simulation loop
    while not rospy.is_shutdown():

        # Get the sensors redings
        sonar = robot.get_sonar()
        pose  = robot.get_robot_pose()
        dist = float(robot.getDistanceToFlag().distance)

        # Write here your strategy..

        # PID controller for speed
        err = dist-d_des
        vt = kp*err + ki*(err_old+err) + kd*(err-err_old)/dt

        # PID controller for angle
        # using poses of robot and flag
        angle_diff = math.atan2(flag1[1]-pose[1], flag1[0]-pose[0])
       angle_err  = angle_diff - pose[2]
       va = 5*angle_err +  ki*(angle_err_old+angle_err) +  kd*(angle_err-
angle_err_old)/dt

        # display some parameters
        print(f"{robot_name} sonar = ",sonar)
        print(f"{robot_name} flag = ",dist)
        print(f"{robot_name} diviation = ",angle,"radian")
```

```
        if sonar < 5.0 and dist > 2.0: # Obstacles avoidance part
            robot.set_speed_angle(2,0.3)
            rospy.sleep(2)


        # Stop robot at small distance
        if dist < 0.1:
          vt = 0
          va = 0

        # store error values for next time step
        err_old = err
        angle_err_old = angle_err

        # Finishing by publishing the desired speed.
        # DO NOT TOUCH.
        robot.set_speed_angle(vt,va)
        rospy.sleep(dt)



if __name__ == "__main__":
    print("Running ROS..")
    rospy.init_node("Controller", anonymous=True)
    run_demo()
```