Control of Mobile Robotics
Spring 2016

Lab 2

Sensors and Actuators

29 February, 2016

Gerardo Sanchez
Adeoluwa Oyenusi

Task Description

The objective of this lab is to perform close loop control using the distance sensors and the robot's motor actuators. This lab consists of 3 tasks.

1. The students are to implement the equations below in order to make the close loop control to signal the correct velocity to move the robot from a distance of 10 inches to a distance of 5 inches from the wall. The values (0.5, 1, 3, 5, and 20) will be used for the proportional gain(K_p).
    a. $u(t) = K_p * e(t)$
    b. $u_r(t) = F_{sat} * (u(t))$
    c. $u_r(t) = F_{sat} * (K_p * e(t))$
    d. $e(t) = r(t) - y(t)$
    e. $u_r(t) = F_{sat} * (K_p * (r(t) - y(t)))$

    - r(t) = desired distance to the goal
    - y(t) = distance from robot to the goal
    - e(t) = distance error
    - K_p = proportional gain
    - u(t) = control signal corresponding to robot velocity
    - f_{sat} = Saturation function
    - $u_r(t)$ = control signal corresponding to saturated robot velocity
2. The robot is to move around the wall while keeping a minimum distance of 5 inches from the wall. The close loop control will be used in this task.
3. The robot is to navigate centrally between walls and make turns at appropriate places. The close loop control will also be used in this lab.

Wall Distance

Video Link: https://www.youtube.com/watch?v=wtj7dGsDHFA

The objective of this task is to have the robot operate using a control system to stop five inches before an obstacle. The desired distance to the goal is, *r(t)*, five inches. The distance is determined from the front short distance sensor. The control system will function under several values of K_p 0.5, 1, 3, 5, and 20.

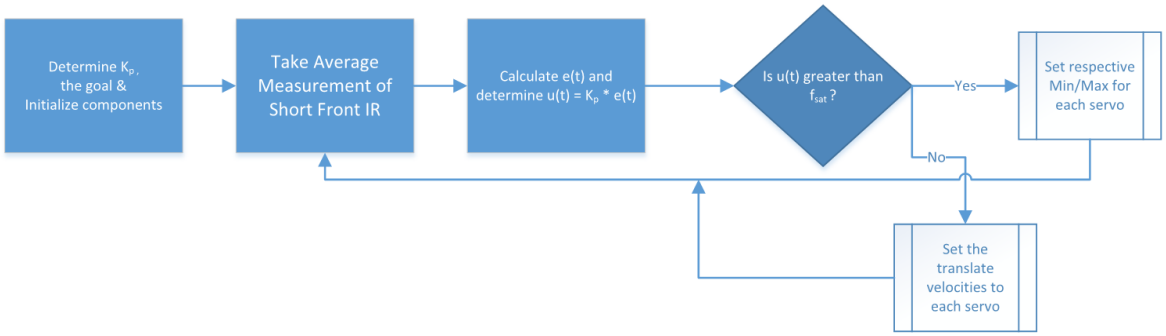
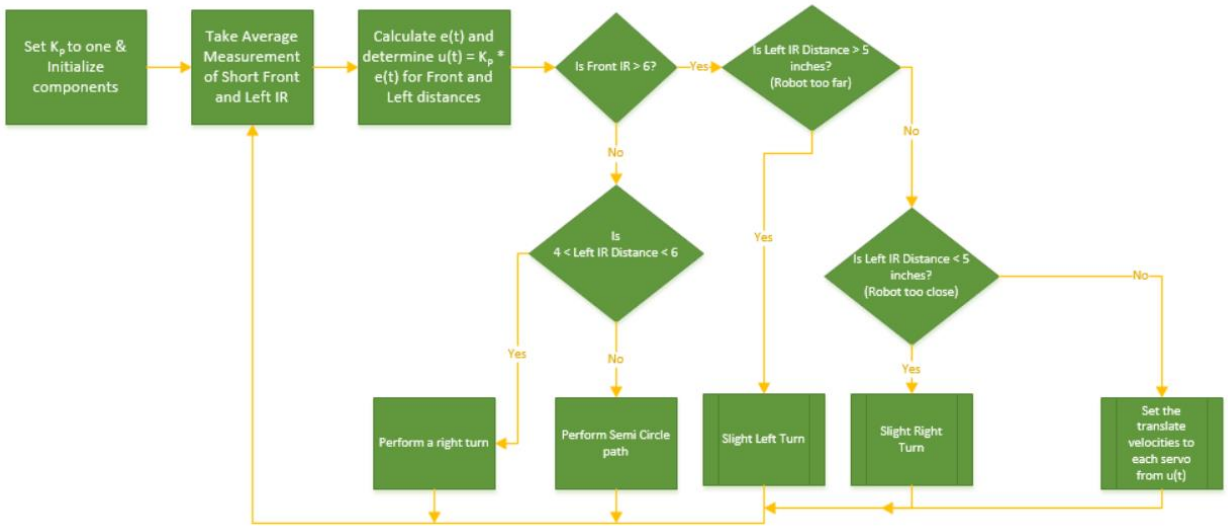

Determine K_p, the goal & Initialize components
Take Average Measurement of Short Front IR
Calculate e(t) and determine u(t) = K_p * e(t)
Is u(t) greater than f_{sat} ?
Yes
No
Set respective Min/Max for each servo
Set the translate velocities to each servo

Figure 1 – Wall Distance Test

Wall Following

Video Link: https://www.youtube.com/watch?v=lQaNdOslliY

Given a wall, the robot will traverse the entirety of the obstacle. It will also maintain a five inch clearance from the perimeter and perform appropriate turns when reaching corners.



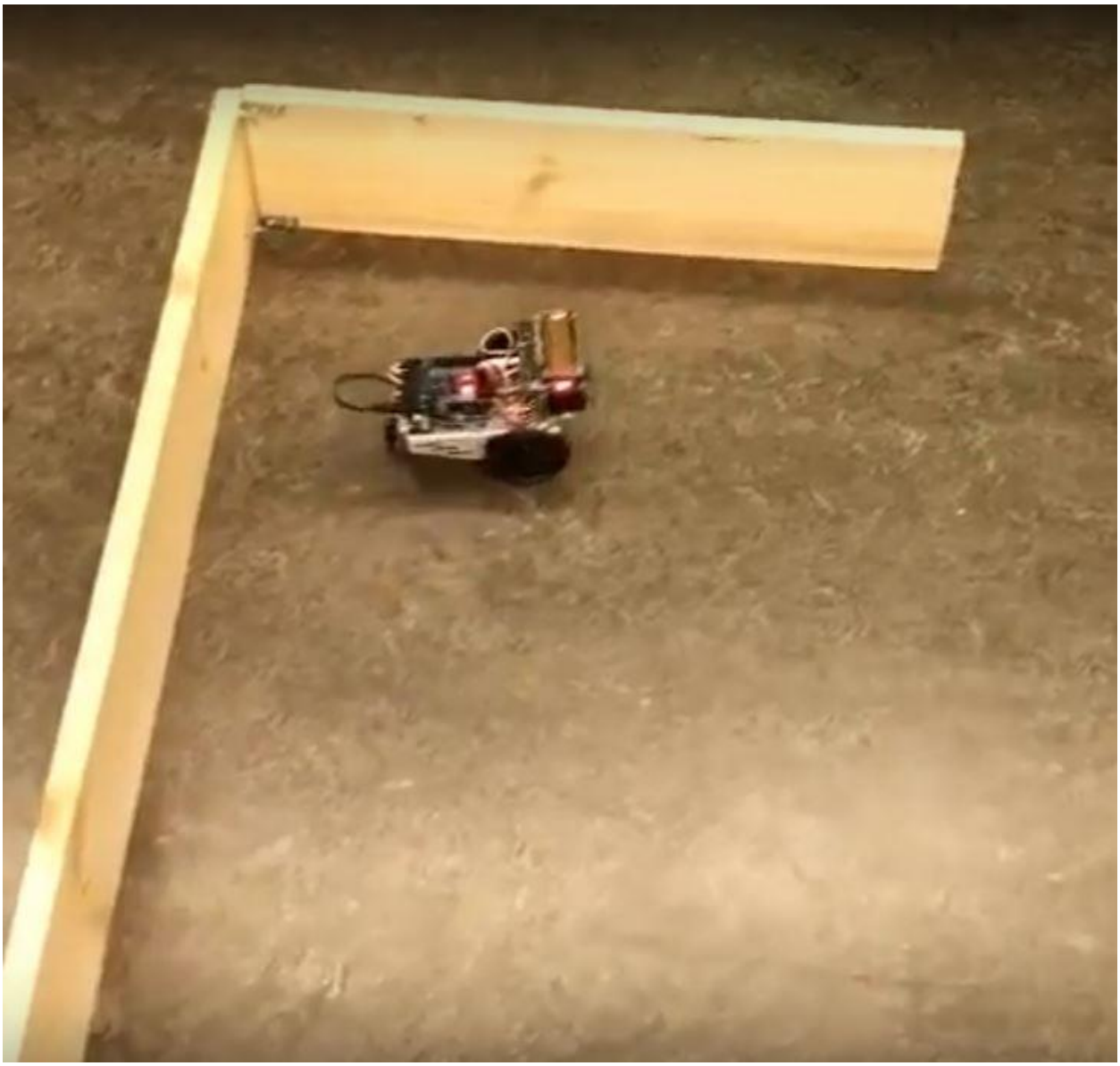
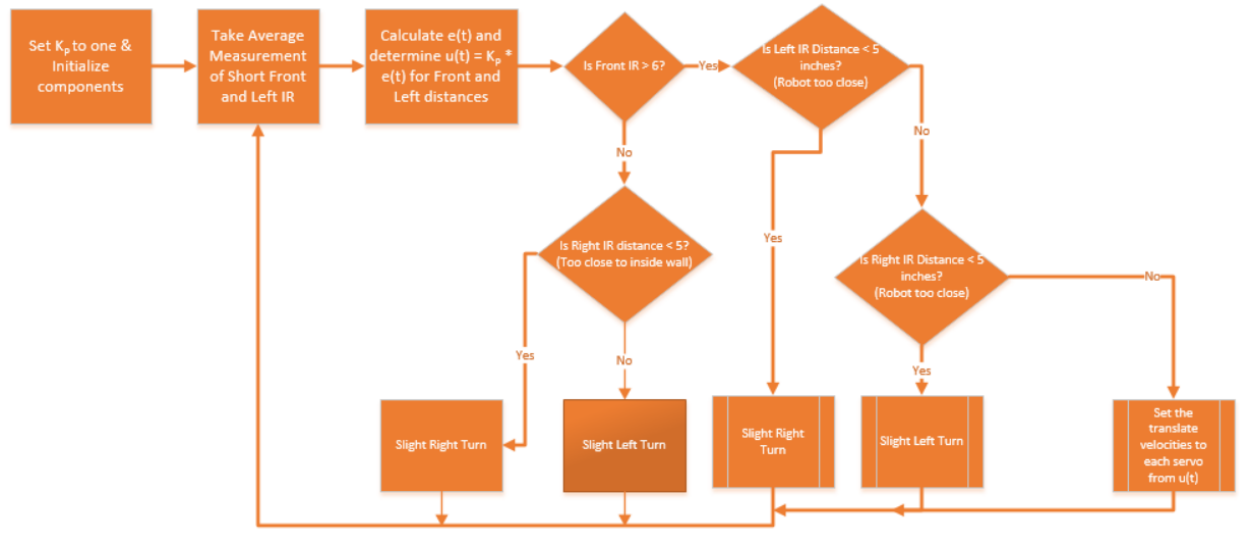


Figure 2 – Wall Following Test

Corridor Navigation

Video Link: https://www.youtube.com/watch?v=6nZaR6pid-s

This scenario matches real world scenarios where the robot needs to navigate through corridors. The robot also uses a closed loop control implementation to read distances and adjust velocities. This loop determines when the robot needs to turn through corners.



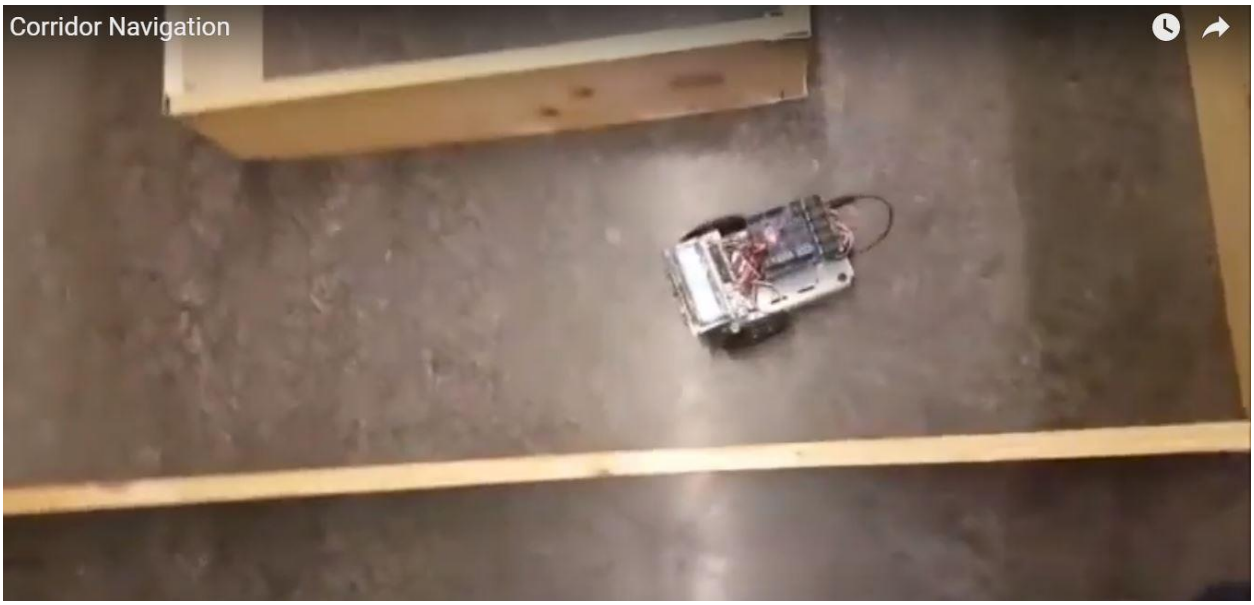


Figure 3 – Corridor Navigation

Code Description

Wall Distance

The code below shows the algorithm used to make the robot stop at 5 inches from the wall. The variable “start” is used to control when the algorithm should be run. First we read the sensor value from the short front sensor and then compute the distance using the “computeDistance_SIR()”, which we used in Lab1. Next, we calculate the error from the desired point by subtracting the distance from the wall from the desired point. Then we calculate the gain by multiplying Kp with the error. We use value from “gain” to control the values going into the

motors. Lastly, we check for saturation so that the values going into the motors will actually fall in the range. The function, cmd_vel(), was implemented in Lab1 and it is used to move the motors.

```
if(start == 1){

      shrtF = average(SFSensor, 5); //short Front

      shrtF = computeDistance_SIR(shrtF);
      //calculate error
      error = set_pt - shrtF;
      gain = Kp * error; //some arbirtary value

      lVel = 90 - gain;
      rVel = gain + 90;

      //check for saturation
      if(lVel > 100)
      {
        lVel = 100;
      }else if(lVel < 90)
      {
        lVel = 90;
      }
      if(rVel < 80)
      {
        rVel = 80;
      }else if(rVel > 90)
      {
        rVel = 90;
      }

      myMover.cmd_vel(lVel, rVel);


      delay(500);
  }
```

WallFollowing

For the Wall Following task, we implemented the algorithm below. We are always reading the distance from both the left sensor and the front sensor. With these values we calculate the error and the gain for each sensor. Next, we check if the distance from the front sensor is greater than 6. If it is, we desire to keep moving forward and if it is not we desire to turn. If the “shrtF” is greater than 6, we check if the robot is too close or too far from the wall and make adjustments. Lastly, if the distance from the wall is equal to 5 inches, we use the gain calculated to feed in values to the motors.

```
void WallFollowing(int Kp)
{
  shrtF = average(SFSensor, 5);
  shrtL = average(SLSensor, 5);
```

```
  shrtF = computeDistance_SIR(shrtF);
  shrtL = computeDistance_SIR(shrtL);

  errorL = SET_PT - shrtL; //error from LEFT
  errorF = SET_PT - shrtF; //error from FRONT
  gainL = Kp * errorL;
  gainF = Kp * errorF;

  if(shrtF > 6) //check front wall to know to keep going forward
  {
    if(shrtL > 5) //too far from left wall
    {
      lVel = 93;
      rVel = 85;
    }else if(shrtL < 5) //too close to left wall
    {
      lVel = 95;
      rVel = 87;
    }else if(shrtL == 5) //align
    {
      lVel = 90 - gainF;
      rVel = 90 + gainF;
    }else
    {
      lVel = 90 - gainF;
      rVel = 90 + gainF;
    }
  } else if(shrtF <= 5)//decision to make right turn
  {
    lVel = 100;
    rVel = 100;
  }


  follow_vel(lVel, rVel);
}

void follow_vel(int lVel, int rVel)
{
  if(lVel > 100) lVel = 100;
    else if(lVel < 90) lVel = 90; //prevent go backward

   //check for saturation in right wheel
   if(rVel < 81) rVel = 81;
        else if(rVel > 90) rVel = 90;

    myMover.cmd_vel(lVel, rVel);
}
```

CorridorNavigation

We followed the algorithm below for the Corridor Navigation task. Similar to the algorithm used in the WallFollowing task, we calculate the distance from the walls from each sensor as well as

their respective gain. Then we check if the robot is close to a wall and if it then the values going into the motors is calculated.

```
void corridorNavigation(int Kp)
{
  shrtF = average(SFSensor, 5);
  shrtL = average(SLSensor, 5);
  shrtR = average(SRSensor, 5);

  shrtF = computeDistance_SIR(shrtF);
  shrtL = computeDistance_SIR(shrtL);
  shrtR = computeDistance_SIR(shrtR);

  errorF = SET_PT - shrtF; //error from FRONT
  errorL = SET_PT - shrtL; //error from LEFT
  errorR = SET_PT - shrtR; //error from RIGHT


  gainF = errorF * Kp;
  gainL = errorL * Kp;
  gainR = errorR * Kp;

  if(shrtF > 6) //check front wall to know to keep going forward
  {
    if(shrtR < 6) //too close to right wall
    {
      lVel = 93 - gainR;
      rVel = 85 + gainR;
    }else if(shrtL < 6) //too close to left wall
    {
      lVel = 94 - gainL;
      rVel = 88 + gainL;
    }else if(shrtL == 5 && shrtR == 5) //align
    {
      lVel = 90 - gainF;
      rVel = 90 + gainF;
    }
  } else if(shrtF <= 6)//decision to make right turn
  {
    lVel = 97 - gainF;
    rVel = 89 + gainF;
  }


  follow_vel(lVel, rVel);
}
```

Conclusion

The purpose of this lab was to make us connect the sensors of our robot with its motors. We were able to control the robot based on the readings of the sensors. We used functions from our previous lab to complete some of the tasks in this lab. For example, we used cmd_vel() from the library we created for lab 1. This function enabled the motors to move. Although we learned

more about our robot during this lab, the main issue we had was getting the robot to not move too far from the wall during the Wall Falling task. We were able to solve this issue, after a series of tests, by reducing the speed of the left motor. As a result, our robot did not go too far away and lose its direction of the front. The lab was good because we were able to control the motors based on information from the sensors.