Memo-ECE425-LAB2

Date: 1/9/2022

To: Professor Berry

Team: Leonardo - Nathan Lee, Shantao Cao

The purpose of this Lab was to get the robot to recognize obstacles and avoid the object using the proper method. For this Lab, we used LEONARDO’s four IR sensors and calibrated them accordingly. The obstacles avoidance behavior used different levels of obstacle avoidance and each level built off each other. The result is that LEONARDO can execute a smart homing function and arrive at a specified location while avoid random obstacles in its path.

**Prelab: State Diagram and Simulation**

The obstacle avoidance behavior was built on levels, each one being built on the previous. The first level was to prevent the robot from hitting obstacles. This was done through aggressive kid and shy kid behaviors. Aggressive kid behavior stops the robot once an obstacle is recognized and shy kid behavior is built on that by being repelled from the obstacle. The planning was done through a state diagram and can be seen in the first figure in Appendix A: avoid obstacle. This behavior was also simulated in Tinkercad (Appendix B) before applying the behaviors on LEONARDO. The next level was to add a wander function with the shy kid behavior to have the robot move around a space while avoiding obstacles (Appendix A: smart wander). The final level was to have the robot go to a set location while avoiding obstacles in its path (Appendix A: smart homing).

**IR Sensor Calibration**

For this lab, we only utilized the four IR sensors on LEONARDO. We calibrated the front IR and the left IR sensors. The back IR uses the same calibration equation as the front IR. The same applies for the left and right IR sensors. Calibration was done through taking data of the analog values to their distance measurement, then linearizing the data and creating a line of best fit (Appendix C)[[1]](#footnote-1). The linearization coefficients were experimentally determined to maximize the R-squares value to 4 decimal places. The IR sensors were calibrated from 1 in to 20 in. The resulting calibration equations:

|  |  |  |
| --- | --- | --- |
|  |  | (1) |
|  |  | (2) |

After implementing the calibration equation for the IR sensors, the distance measurement is discrete to the nearest inch with an accuracy of 1 inch.

The IR sensors was enough for LEONARDO to sense its terrain and the use of the additional sonar sensors would inhibit its behavior. For the sonar sensors, we have tried using the NewPing arduino library and it gave us accurate distance readings. During implementation of the sonar sensors for the shy kid behavior, one of the two sonar sensors would trigger and give us false obstacle readings. The IR sensors proved to be enough for this lab.

**Provided Move Function**

In move1(), we observed that increasing step time decreases running speed. This is expected because a longer step time would run the wheel speeds slower.

In move2(), swapping runSpeedToPosition() to runToNewPosition() resulted in the right wheel running and stopping then the left wheel running and stopping. This is caused by the runToNewPoistion function blocking code and preventing the next line of code to run before the command has been fully executed while the runSpeedToPosition function is not blocking and runs the command while moving on the next line.

In move3(), we changed the runSpeedToPosition function to the run function. The stepper motors only move one step after we upload the code using the run function. The run function only steps the motors one step per function call.

The move4 function was not behaving as described in the comment. The comment describes the function as making two wheels run at two different speeds, but the robot wheels only take one step for each call of the move4 function. It is hard to confirm that the wheels were moving at different speeds because the wheels do not rotate. We replaced the runToPosition functions to the runAtSpeedToPosition function and observed the wheels oscillate and step back and forth.

The move5() function, the wheels run at a set speed and there was no difference in speeds between the wheels.

**Basic Movement**

Our robot, Leonardo, has a wheel diameter of 3.375 inches. Each step is 1.8°, so the basic movements were calculated using the step per degree and wheel diameter shown in Table 1

|  |  |
| --- | --- |
| Function | Steps |
| forward() | 1811 steps / 2 feet |
| spin() | 504 steps / 90° |
| pivot() left | 888 steps / 90° left |
| pivot() right | 1058 steps / 90° right |

**Table 1**: The recorded steps it takes to execute the basic functions

**Encoder**

The goal for the encoders is to close loop the motors and help compensate for odometry error. In practice we have found that the motors do not skip steps so using a correction factor is more accurate than the encoders. The motor step is more accurate than the encoder ticks because each motor step is 1.8° while each encoder tick is about 19°, shown in Table 2. The major odometry error is from wheel slippage on the floor and cannot be fixed using encoders.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Rotation | Left Ticks | °/Tick | Right Ticks | °/Tick | **Table 2**: These are the corresponding encoder values for each partial rotation for the left and right motor. |
| 90° | 5 | 18°/Tick | 5 | 18°/Tick |
| 180° | 10 | 18°/Tick | 8 | 22.5°/Tick |
| 360° | 20 | 18°/Tick | 18 | 20°/Tick |
| 720° | 40 | 18°/Tick | 36 | 20°/Tick |

**Circle and Figure 8**

For implementing the circle function, we modeled the robot as a rigid body rotating about the circle’s center with a constant velocity set in the function. From this model, we found the wheel speeds and distances each wheel would cover. The wheel speeds were set, and the distances were converted to steps for the motors. The robot simulation is based on time for rotation while Leonardo works better with motor steps. A correction factor was then added to the speed and steps to help account for wheel slip and other odometry factors. The results of the circle can be seen in Table 3. The figure 8 function uses the circle function in the positive direction then negative direction.

|  |  |  |
| --- | --- | --- |
| Run  Number | Circle  Diameter (36”) | **Table 3**: After adjusting the correction factors, the error between real diameter and expected diameter is within 2 inches. |
| 1 | 32 |
| 2 | 36 |
| 3 | 40 |
| 4 | 35 |
| 5 | 37 |

**Go To Angle, Go To Goal, and Square**

The goToAngle function uses the same model and strategy as the circle function, but the robot pivots about the inner wheel. The error in goToAngle() can be found in Table 4. The goToGoal function converts the cartesian coordinates to polar coordinates. The angle is calculated using the atan2 function and distance is calculated using the pythagorean theorem. The polar coordinates make goToAngle() implementation easier by running goToAngle() with the calculated angle and forward() with the calculated distance. The square function was planned to use the goToGoal function in the pseudo code, but a combination of goToAngle() and forward() made implementation easier.

|  |  |  |  |
| --- | --- | --- | --- |
| Angle Set | Angle Executed | Error | **Table 4:** The goToAngle function was tested using multiple angles. The error are listed for the tested angles. |
| 90° | 90° | 0° |
| -90° | -90° | 0° |
| 60° | 63° | 3° |
| -60° | -61° | 1° |
| 45° | 45° | 0° |
| -45° | -45° | 0° |
| 10° | 14° | 4° |
| -10° | -12° | 2° |
| 30° | 27° | -3° |
| -30° | -30° | 0° |

**Library Functions**

Table 5 explains the example code provided in the AccelStepper Library.

|  |  |
| --- | --- |
| **Example Code** | **Description** |
| Block | Sets a new target position and then waits for the steppers to achieve it. |
| Bounce | Moves back and forth a certain distance. |
| MultipleSteppers | Runs multiple stepper motors back and forth. |
| MultiSteppers | Combine multiple steppers into one object and calls the stepper all at once. |
| Overshoot | Sets a new target position and waits until the stepper has achieved it. |
| ProportionalControl | Runs the stepper motor at a speed proportional to the voltage input. |
| QuickStop | Runs the stepper motor at full speed and stops it quickly with the set acceleration. |
| Random | Runs the stepper motor with random changes in speed, position and acceleration. |

**Table 5**: Provided example code with the AccelStepper Library.

**Appendix A: Pseudo code for Advance Function**

moveCircle(int diam, int dir)

* initialize speed(v) to 10 // the default center linear speed is 10 (units TBD)
* initialize radius(r) to diam/2
* initialize rotational speed(w) to v/r
* initialize runtime(t) to 2\*pi/w
* If the direction is positive (counter clockwise):
  + find inner wheel linear speed from w: vinner = w \* (r-dleft) <- dleft is distance from center bot to left wheel and vleft is linear speed of left wheel
  + find outer wheel linear speed from w: vouter = w\*(r+dright) <- dright is distance from center bot to left wheel and vright is linear speed of left wheel
  + Set left motor speed to vinner speed
  + Set right motor speed to vouter speed
* Else direction is negative (clockwise):
  + find inner wheel linear speed from w: vinner = w \* (r-dright)
  + find outer wheel linear speed from w: vouter = w\*(r+dleft)
  + Set left motor speed to vouter speed
  + Set right motor speed to vinner speed
* Delay for t time
* Stop both wheels

moveFigure8(int diam)

* Run moveCircle with diam and positive direction
* Run moveCircle with diam and negative direction

go-to-angle(int angle)

* initialize speed(v) to 10
* initialize radius(r) to dleft(or d right) dleft is distance from center bot
* initialize rotational speed(w) to v/r
* initialize runtime(t) to (angle/360)\*2\*pi/w
* If the angle is positive:
  + Run Spinleft at speed 10
  + Delay for t time
  + Stop both wheels
* Else angle is negative:
  + Run Spinright at speed 10
  + Delay for t time
  + Stop both wheels

go-to-goal(int x, int y)

* initialize speed(v) to 10
* Initialize thetad(thetad) to atan2(y,x)
* Initialize distance(d) to sqrt(y^2+x^2)
* initialize runtime(t) to d/v
* Run go-to-angle(thetad)
* Run forward at speed v
* Delay for t time
* Stop both wheels

moveSquare(int side)

* initialize speed(v) to 10
* Run go-to-goal(side, 0)
* Delay for 1s
* Run go-to-goal(0, side)
* Delay for 1s
* Run go-to-goal(0, side)
* Delay for 1s
* Run go-to-goal(0, side)
* Delay for 1s

**Appendix B: Simulation Code**

#include <Servo.h>

Servo leftservo;

Servo rightservo;

const int dleft = 50;

const int dright = 50;

void setup() {

leftservo.attach(9);

rightservo.attach(10);

//move forward fast

leftservo.write(170);

rightservo.write(10);

delay(5000);

// circle

circle(100,1); //works better for numbers closer to 100

// figure 8

figure8(60);

// square

square(1);

//go to goal

gotogoal(-5,-1);

}

void circle(int dia, int dir){

int v = 100;

int r = dia/2;

float w = v/r;

int t = (0.1858\*dia+3.2542)\*1000; //number was found by trendline assuming linear

if (dir > 0){

int vinner = map(w\*(r-dleft),0,v,90,180);

int vouter = map(w\*(r+dright),0,v,90,0);

leftservo.write(vinner);

rightservo.write(vouter);

} else if(dir < 0){

int vinner = map(w\*(r-dright),0,v,90,0);

int vouter = map(w\*(r+dleft),0,v,90,180);

rightservo.write(vinner);

leftservo.write(vouter);

}

delay(t);

stop();

}

void figure8(int dia){

circle(dia,1);

circle(dia,-1);

}

void square(int l){

for (int i = 0; i < 4; i++){

leftservo.write(170);

rightservo.write(-180);

delay(1000\*l);

gotoangle(3.1415/2);

}

stop();

}

void gotogoal(int x, int y){

float theta = atan2(y,x);

int l = sqrt(sq(x)+sq(y));

gotoangle(theta);

delay(1000);

leftservo.write(170);

rightservo.write(-180);

delay(1000\*l);

stop();

}

void gotoangle(float angle){

int t = sqrt(sq(angle\*300\*5.7));

if (angle > 0){

leftservo.write(-180);

rightservo.write(-180);

delay(t);

}

else if (angle < 0){

leftservo.write(180);

rightservo.write(180);

delay(t);

}

else {

delay(1000);

}

stop();

}

void stop(){

//stop moving

leftservo.write(90);

rightservo.write(90);

}

void loop() {

}

**Appendix C: Leonardo - Bipedal Robot with Thrusters**



Our robot is named after LEONARDO, the bipedal (2 legged) robot that utilizes thrusters for locomotion developed by Caltech. LEONARDO is an acronym for LEgs ONboARD drOne, or LEO for short. The project was started during the summer of 2019 and was published on October 6, 2021.

LEO combines the benefits of propeller based thrusters and leg joints to tackle complex real world terrain in a similar method humans would. The benefit with LEO is that the use of the thrusters lets LEO overcome more difficult terrain humans would not be able to overcome.

More information can be found in the source from Caltech’s News “LEONARDO, the Bipedal Robot, Can Ride a Skateboard and Walk a Slackline” or Science Robotics’ paper “A bipedal walking robot that can fly, slackline, and skateboard”.

Source: <https://www.caltech.edu/about/news/leonardo-the-bipedal-robot-can-ride-a-skateboard-and-walk-a-slackline>

1. https://acroname.com/articles/linearizing-sharp-ranger-data [↑](#footnote-ref-1)