Artificial Intelligence in Mobile Robots
Lab 1 - Getting started

Lab Assistant: Ali Abdul Khaliq

September 15, 2016

Lab completed: 2016-09-05

Lab Students: Name - PID - Email
Tobias L - 870603-6657 - tobiaslindwall@gmail.com
Özgun M - 920321-2379 - ozgun.mirtchev@gmail.com

1 Overview

In this lab the students were asked to get familiar with Linux and to learn about how the ePuck robot works. There were 13 total tasks to be done, with 5 optional ones. The objective of the tasks was trying to design functions to make the ePuck move and calibrate the constant values to make sure the functions are performing according to what is the expected output values. In example, how far the robot runs and how much it turns. In the optional tasks the objectives was mostly to compare different functions to see how much they differed between each other and to compare how the ePuck behaved.

2 Tasks

2.1 Familiarizing with Linux

Since both of the lab students were already familiar with Linux and the objectives of this task were basic, this step was mostly skipped.

2.2 Read and download material

A close look into the documentation of ePuck and the code was looked and analyzed before moving on.

2.3 Run the sample program on the ePuck

In the sample program the ePuck was moving forward and turning.

2.4 Run the sample program on the simulator

The simulator runs in relative positioning so SetTargetSteps are going to be different in the simulator and different for the real ePuck. The real ePuck is in absolute positioning meaning it will travel the correct distance for SetTargetSteps compared to the simulator. One thing that was noticed in the simulator was that the movement was faster than the real ePuck, because of the relative and absolute differences. The simulator did however do slower IR-readings than the real ePuck.

2.5 SetTargetSteps()-function

The function takes left and right values for respective wheel of the ePuck, which in turn activates the motors. A value of 1000 to both left and right gives 1000 pulses per 10 msec. SetTargetSteps(2000,3000) makes the right wheel turn for 2 rotation and the left for 3 total rotations. It continues to turn one more rotation after the right one has finished, thus turning the car to a leftward direction.

2.6 SetSpeed()-function

SetSpeed makes the ePuck travel the same distance in the simulator and in the real environment. Sleep is required to stop the program from continuing to the next line of code to allow the vehicle to travel in said speed. Stop is needed to make the ePuck stop so it doesn't run forever. SetSpeed reads values of right and left like SetTargetSteps. The minimum/maximum which can be passed to the function is -1000 /1000. One observation when running SetSpeed close to 0 was that it was jerking around which caused it to turn unintendedly. When the SetSpeed-values was set to 300, 400 for a short time the robot didn't move at all, at least from a visual perspective. It might've moved a very short distance but it wasn't noticeably to the human eye. This is because the time intervall for the robot to move. When it received the message to initiate SetSpeed, it received a Stop-message immediately after.

2.7 Moving the robot forwards and backwards with function SetTargetSteps

Here we made a simple for loop, which loops the function inside five times. It reverses by changing the direction values by multiplying by -1.

Code can be found in Appendix: Code1.

2.8 Moving the robot forwards and backwards with function SetSpeed

The same method as the previous one was used for this one.

Code can be found in Appendix: Code2.

2.9 (Optional) Move-function to move the robot a certain direction

To make the ePuck move a given distance in millimeter, one important thing to know was how many steps it took for the ePuck wheel to rotate one revolution. As mentioned before, one revolution of the wheel was 1000 steps, the circumference of the wheel was 129.05 mm. Amount of steps per mm was given by: 1000 steps / 129.05 mm ~= 7.75 steps/mm. By measuring the travelled distance of the ePuck and monitoring the values it produced, the end-result was calibrated to 7.71 steps/mm.

Code can be found in Appendix: Code3.

2.10 (Optional) Turn-function to turn the robot a certain degree from where it's facing

To get how many steps was required for the ePuck to turn 1 degree, the circumference of the ePuck was used. The given diameter of the ePuck was 53 mm. To get the circumference: 52.55 mm * PI = 165.0 mm. To get the number of steps required for one degree, the steps from the previous task was used: 165.0 mm * 7.71 steps ~= 1272.20 steps. Steps per degree: 1272.20 steps / 360 degrees = 3.53 steps / degree.

This was later calibrated to 3.63.

Code can be found in Appendix: Code4.

2.11 (Optional) Move-function version 2

This function is supposed to do the same as the previous function Move, but using the given function SetSpeed instead of SetTargetSteps. A "busy wait" loop was used instead of sleep to keep the wheels going until the condition to travel a certain distance was fulfilled. When this was fulfilled, the function Stop was called. This Move function was less precise than the previous one.

Code can be found in Appendix: Code5.

2.12 (Optional) Move-function 3 - Not completed.

2.13 (Optional) Simple IR-program.

This program makes the ePuck detect which of the sensors are receiving low light and activates the corresponding led to the IR-sensor number.

Code can be found in Appendix: Code6.

3 Conclusions

We had limited experience with Linux before we started this course, but doing this lab and programming in this environment has increased our understanding and experience some.
At first we had some problem using the rfcomm command to connect to the ePuck through bluetooth. This was solved by using the ePuck's mac adress as final argument in rfcomm (rfcomm connect rfcomm2 XX:XX:XX:XX:XX:XX).
Designing the movement functions for the ePuck was interesting because we had to analyse data from the ePuck and adjust our source code according to the requirements. It was also interesting to calibrate factors such as "steps per mm" or "steps per degree" and see improvements.
We are satisfied with our results, but if we were to improve it, we could do more tests and measurements of i.e. the wheels or maybe axis deformations and implement compensations.

Appendix

Code1:

```
int direction = 500;
for(i = 0; i < 5; i++) {
   SetTargetSteps(direction, direction);
   Sleep(500);
   direction *= -1;
}
```

Code2:

```
int direction = 500;
for(i = 0; i < 5; i++) {
   SetSpeed(direction, direction);
   Sleep(1000);
   direction *= -1;
} Stop();
```

Code3:

```
// 1 mm = 7.71 steps
void Move(double mm) {
   float StepsPermm = 7.71;
   int steps = mm * StepsPermm + GetSteps().l; //
      GetSteps().l (!)
   printf("Moving %lf millimeters\n", mm);
   printf("Setting steps: %d\n", steps);
   SetTargetSteps(steps, steps);
}
```

Code4:

```
// 1 degree = 3.63 steps
void Turn(double degrees) {
   float StepsPerDegree = 3.63;
   int steps = degrees * StepsPerDegree + GetSteps().l;
   printf("Turning %lf degrees\n", degrees);
   printf("Setting steps %d:\n", steps);
   SetTargetSteps(steps, -steps);
}
```

Code5:

```
void Move2(double mm) {
   float StepsPermm = 7.71;
   int initialSteps = GetSteps().l;
   float target = GetSteps().l + mm * StepsPermm;
   SetSpeed(400, 400);
   // "Busy wait" until target is reached...
   while (GetSteps().l * 1.0 < target) {
      printf("%d steps!\n", GetSteps().l);
   }
   Stop();
   printf("Final steps: %d\n", GetSteps().l);
   float movedmm = (GetSteps().l - initialSteps) /
       StepsPermm;
   printf("Moved %lf mm!\n", movedmm);
}
```

Code6:

```
void IRDetector() {
   Sensors ir; // get sensors voltage unsigned int
       (sensor)
   bool led[8] = {0,0,0,0,0,0,0,0}; // eight leds
   SetRingLED (led);
   int i, j;
   for (i=0;i<1000;i++) {
      ir = GetIR();
      printf("IR values: %4d, %4d, %4d, %4d, %4d, %4d,
          %4d, %4d\n",
         ir.sensor[0], ir.sensor[1], ir.sensor[2],
         ir.sensor[3], ir.sensor[4], ir.sensor[5],
         ir.sensor[6], ir.sensor[7]);

      for (j=0; j<8; j++) {
         if (ir.sensor[j] > 1000)
            led[j] = 1;
         else
            led[j] = 0;
      }
      SetRingLED(led);
   }
}
```