compat=1.12

Laboration 3: Differential Drive with WiFi control

Sensors and Sensing

Benny Frost, Tom Olsson

January 17, 2016

All code for this exercise can be found at
`https://github.com/tgolsson/ros-arduino-wifi`

Contents

Code listings

List of Figures

List of Tables

1 Motivation and theory

The goal of this project was to implement a differential drive robot based on the Robot Operation System [ROS] on Arduino, which can be controlled using a WiFi connection. While a differential drive is easy to implement and test using a wired connection, most realistic scenarios will require wireless operation. As ROS is very common in the scientific community, and Arduino is a cheap prototyping platform in comparison to prebuilt robots, this could allow a much larger freedom in robot design.

The goals are:

⇒ To construct a robot with two powered wheels

⇒ Setup the two motors with a differential drive controller

⇒ Implement WiFi communication between PC and Arduino

⇒ Tune the odometry

1.1 PID with feedback

PID in the name PID-controller is short for *Proportional-Integral-Derivative*-controller. As this implies, the controlling signal is based on a proportion of the current error, the previous error, and the rate of change of the observed error. Sometimes, the error is augmented to include a part of the previous output value. In the case of velocity control it can be used to counteract overshoot if the observable value does not react instantly to output value. The mathemathical formulation of this is shown in (1)-(2).

Let:

$e(t)$	be some error measurement between current state x(t) and preferred state r(t)
K_p, K_i, K_d	be the respective weights for the proportional, integral and derivate terms
$u(t)$	be the output signal at time t

Then:

$$e(t) = r(t) - x(t) - u(t-1) \tag{1}$$

$$u(t) = K_\mathrm{p} \cdot e(t) + K_\mathrm{i} \cdot \int_0^t e(\tau) \cdot \mathrm{d}\tau + K_\mathrm{d} \cdot \tfrac{\mathrm{d}e(t)}{\mathrm{d}t} \tag{2}$$

This formulation of the PID controller borrows from feed-forward open-loop control by partially ignoring the current measurements[1]. The controller is shown in fig. 1.

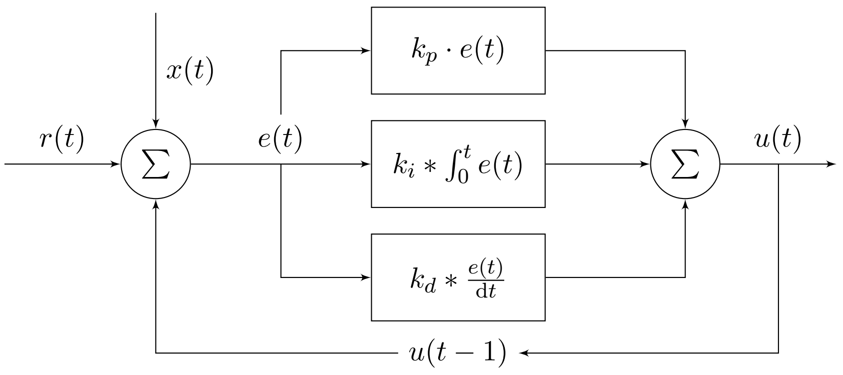

Figure 1: PID controller with feedback term.

[1]An open loop controller has no sensory information: it has only state and model.

Another modification of PID controllers to improve motor control is the addition of an output deadband or *stiction*[2]. Stiction is the static cohesion threshold that needs to be overcome in order to bring an object from rest when in contact with other objects. In the case of a motor, this can be seen as the minimum `PWM`-value where the shaft starts turning consistently.

1.2 Communication rates

As the `Arduino ros_lib` implementation is based on the premise of serial communication, some things need to be taken into consideration. No matter whether operating over the USB port or a direct `TX,RX` connection a serial protocol is always synced, and has a more or less constant rate of information, and has predictable packet sizes.

TCP/IP on the other hand - which is the wireless protocol for a sustained server/client connection - is not synced, nor does it make any guarantee on any communication speeds. The only guarantee is that the data will arrive at some point, and that it will be possible to retrieve the packages in proper order. Compared to serial it also makes no guarantee on packet size; and may sometimes try to limit the number of packages sent by collating many small packets into larger ones using f.ex. Nagle's algorithm.

This discrepancy means that one cannot rely on data arriving regularly, and must make sure that the implementation can handle the problems that can occur. These problems could be a message being broken up into multiple parts, or only recieving half the message, a connection dropout, etc.

1.3 Differential Drive

A differential drive is a kinematic model for two individually controlled wheels used to control both speed and rotation by altering the relative speed between the wheels. The controller implemented here is an inverse kinematic model; i.e., the desired output is given and then the kinematic model is based to give these output parameters.

Let:

v, ω	be the desired linear (m/s) and angular velocities (rad/s)
L	be the distance between the two wheels
r	be the radius of the robots wheels
k	be the number of encoder ticks per shaft rotation
u_l, u_r	be the left and right encoder speeds

Then:

$$u_l = \frac{2\pi r}{k} \cdot \begin{cases} \left(\frac{v}{\omega} - L\right) * \omega & if \omega \neq 0 \\ v & otherwise \end{cases} \tag{3}$$

$$u_r = \frac{2\pi r}{k} \cdot \begin{cases} \left(\frac{v}{\omega} + L\right) * \omega & if \omega \neq 0 \\ v & otherwise \end{cases} \tag{4}$$

1.4 Odometry

The kinematic model for differential drives can be used to estimate the robots current position relative to the starting position. The equations for updating the robots position are shown in eq. (5)-(13) on the following page.

[2]A portmanteau of *static friction*

Let:

L	be the distance between the two wheels
r	be the radius of the robots wheels
k	be the number of encoder ticks per shaft rotation
$\hat{u}_l, \hat{u}_r$	be the smoothed left and right measured encoder speeds (*ticks/second*)
x, y, θ	be the position and rotation of the robot in a 2D coordinate system

Then:

$$l_{left} = \frac{2\pi r}{k} \cdot \hat{u}_l \tag{5}$$

$$l_{right} = \frac{2\pi r}{k} \cdot \hat{u}_r \tag{6}$$

$$\Delta = \frac{(l_{right} - l_{left})}{L} \tag{7}$$

$$d = \frac{l_{right} + l_{left}}{2} \tag{8}$$

$$\Delta_x = \begin{cases} d \cdot \cos\left(\frac{\Delta}{2}\right) & if\, |\Delta| < \epsilon \\ \frac{d}{\Delta} \cdot \sin(\Delta) & otherwise \end{cases} \tag{9}$$

$$\Delta_y = \begin{cases} d \cdot \sin\left(\frac{\Delta}{2}\right) & if\, |\Delta| < \epsilon \\ \frac{d}{\Delta} \cdot \big(1 - \cos(\Delta)\big) & otherwise \end{cases} \tag{10}$$

$$x_t = x_{t-1} + \Delta_x \cdot \cos(\theta_{t-1}) - \Delta_y \cdot \sin(\theta_{t-1}) \tag{11}$$

$$y_t = y_{t-1} + \Delta_y \cdot \sin(\theta_{t-1}) + \Delta_y \cdot \cos(\theta_{t-1}) \tag{12}$$

$$\theta_t = \theta_{t-1} + \Delta \tag{13}$$

2 Implementation

The purpose of this project is to build a robot with two drive wheels. This robot shall use an Arduino Due controller board with a differential drive controller for these wheels, and communicate to a master node using WiFi and TCP/IP. Part of the project is also measuring and tuning the accuracy for the robots odometry.

2.1 Hardware and environment

The project used an *Arduino Due* microcontroller [1], with the *Arduino Motor Shield R3* [2]. These are programmed using Serial-over-USB; with the dedicated IDE. The version of the IDE used is 1.6.5. The project also includes usage of the *Robot Operating System* [ROS], version *Indigo Igloo*. Two WiFi modules were tested during the project, the official *WiFi Shield R3* [3] as well as the *ESP8266* [4].

The motors used are the *Micro Motors RHE158 75:1 12V DC* [5].

2.2 Physical design

The main body of the robot is built from aluminum plating and profiles to get a robust robot. The circuit boards are attached to Plexiglas which then are attached to the main body. This is done to get a good insulation from the main body. The Plexiglas pieces are attached with Velcro so it will be easy to detach them if necessary. At front the two motors with the driving wheels was mounted and two passive wheels where mounted at the back of the robot. The passive wheels do not swivel and there was a concern if this would work. Testing showed that the two electrical motors were too weak to be able to turn the robot with this setup. This is probably because the friction against the floor is too high. A centered swivel wheel was therefore mounted at the rear of the robot. The robot is shown in fig. 2 on the next page.

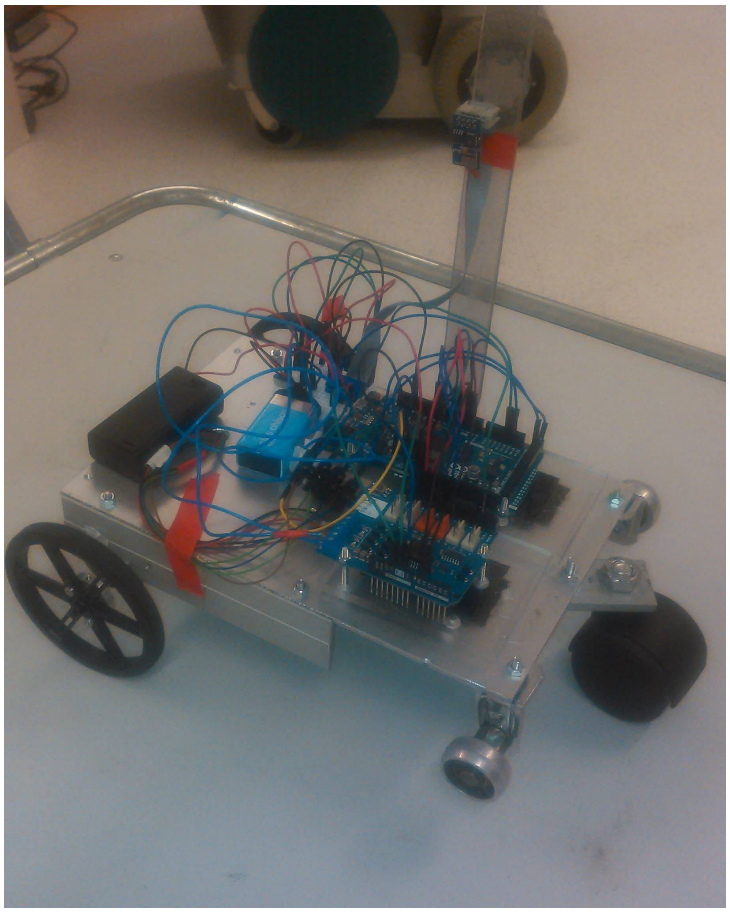

Figure 2: The robot. The centered swivel wheel is at the back of the robot. Motor battery pack not mounted for better viewing.

For controlling the robot, an Arduino Due microcontroller is used with an Arduino Motor-shield, and for communication an Arduino WiFi shield is used. These two shields are made to sit on top of the Arduino Due. However, this means that the WiFi shield and Motor shield will also sit on top of each other. Since the WiFi shield and the Motor shield use the same I/O pins of the Arduino Due it was decided that only the WiFi shield should be mounted on top of the Arduino Due and the Motor shield should be mounted next to the main board and be connected to the Arduino Due with wires, and by doing this I/O pins could be rerouted to be used for the Motor shield. Fig. 3 on the following page shows how the Arduino Due, Arduino Motor shield and ESP8266 are connected.

2.3 WiFi communication

During setup and configuration of the Arduino WiFi shield it was obvious that the communication was very slow, taking almost two seconds from sending data to reaching the destination, which was documented by others online [6, 7]. Similarly, it was discovered that the largest package that could be sent to the Arduino WiFi card is 92 bytes. If the package is larger there will be a silent failure. With all this it was decided to use another WiFi card. We decided to use a very popular WiFi card, ESP8266.

The ESP8266 is a standalone microcontroller with built-in WiFi support [4]. This card is often recommended instead of the official WiFi card both because of its cost as well as superior performance, which is the reason we use it. Similarly to the normal arduino boards it comes with its own toolchain, and therefore runs the code on its own in a separate process from the main Arduino board. This allows the card to be setup to handle I/O asynchronously from the

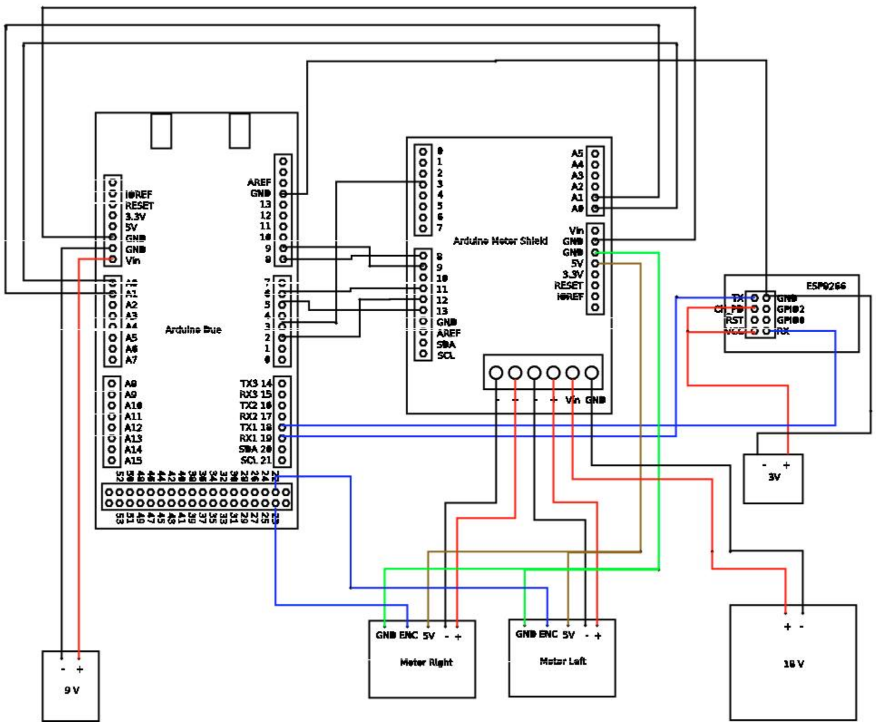

Figure 3: The wiring diagram for the robot's electronics

main control loop which is a big performance gain. The final communication setup is shown in fig. 4.

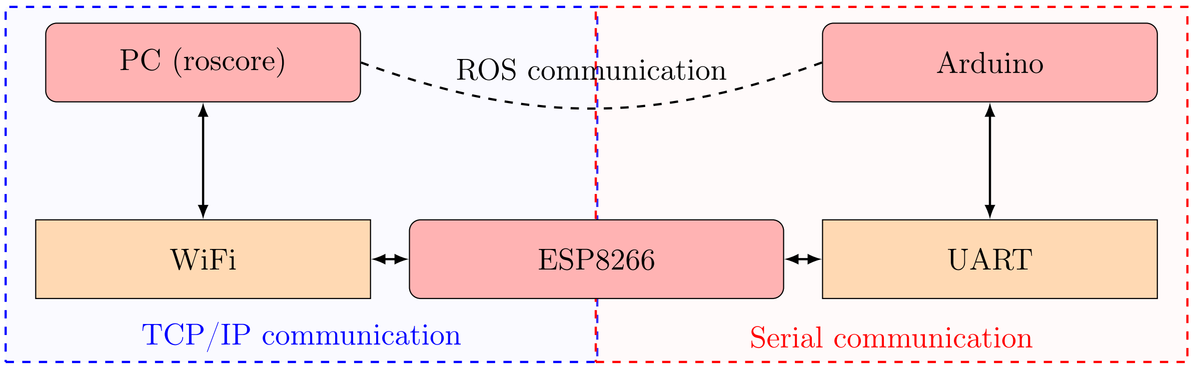

Figure 4: System view of communication

2.4 Controller node

The node used to steer the robot was developed by Husqvarna for their lawn mower *Auto Mower 320* and used for research at Örebro University. This node fulfilled the requirements that was needed for this project. It is a Python script that reads from the keyboard and publishes a `geometry_msgs::Twist` message on the ROS bus. The message uses `linear.x` for forward speed and `angular.z` for angular speed.

2.5 Passthrough node

The WiFi node is also very simple. During startup it connects to a dedicated WiFi router, and then connects to a hard-coded server IP. In the main `loop` function it reads from the serial and writes to the WiFi; as well as from the WiFi to the serial as was shown in fig. 4. To make it more reliable the node checks both network and server connection each iteration, and attempts to reconnect if any issues are detected.

2.6 Arduino node

This node has two PID controllers with their own K_P, K_I and K_D values, to set the velocity of the two wheels. It is necessary to have two PID controllers since the motors appear to have slightly different performance.

On major problem when implementing PID controllers to control the electrical motors is the deadband of the motors. The output from the controller is 0-4096 and between 0-700 nothing happens. To solve this; if the output is between 0-10 the output is set to 0, and if the output is between 11-700 the output is set to 700, else the output is clamped to be in between 700-4096.

The input signal from the encoders is smoothed with a 330ms sliding window, that means to get a velocity change to be fully visible it takes around 330 ms. This also means there is a delay that the PID controller doesn't know about, and it will therefore output a value without getting a signal back that something happens. This would cause the controller to increase it's output, and in this way it will overshoot by a large margin. To counteract this, a fraction of the output is fed back into the input as a promised velocity that will happen, so the controller will not ramp up to high output and therefore not overshoot.

For the communication with ESP8266 the Serial1 (`TX1,RX1`) is used instead of the usual Serial/USB (`TX0,RX0`). With this setup it is still possible to send debug data to the Arduino IDE.

This node subscribes to `/robot_0/velocity_commands` (`geometry_msgs::Twist`) to get the commands. It advertises its pose on `/robot_0/odometry` (`geometry_msgs::Pose2D`).

3 Results

3.1 PID tuning

3.2 Odometry accuracy

References

[1] Arduino LLC, "Arduino - arduino due." `https://www.arduino.cc/en/Main/ArduinoBoardDue`. (Last Visited on 01/10/2016).

[2] Arduino LLC, "Arduino - arduino motor shield." `https://www.arduino.cc/en/Main/ArduinoMotorShieldR3`. (Last Visited on 01/10/2016).

[3] Arduino LLC, "Arduino - arduino wifi shield." `https://www.arduino.cc/en/Main/ArduinoWiFiShield`. (Last Visited on 01/10/2016).

[4] NURDSpace, "Esp8266 - nurdspace." `http://nurdspace.nl/ESP8266`, November 2014. (Last visited on 01/10/2016).

[5] Reductor Motor, "Rh158.12 , rh158.24 micromotors." `http://www.reductor-motor.com/eng-micRH158.htm`. (Last Visited on 01/17/2016).

[6] MattS-UK, "Arduino wificlient (tcp)." `http://mssystems.emscom.net/helpdesk/knowledgebase.php?article=51`, November 2013. (Last visited on 01/10/2016).

[7] Group discussion, "Wifi shield tcp to webserver is very slow." `http://forum.arduino.cc/index.php?topic=123824.0`, September 2012. (Last visited on 01/10/2016).