Robot Platform Controller with Arduino Mega 2560

Florian Polster
E-Mail: fpolster@uni-koblenz.de
Team Homer, Universität Koblenz-Landau
http://homer.uni-koblenz.de

Abstract—This paper shows an easy way to control a robot platform with an Arduino Mega 2560. We will use the rosserial package to integrate the presented system into ROS. This will be shown with the Pioneer 3-AT platform, but can be used for other platforms as well.

I. HARDWARE

A. Hardware

To control the pioneer 3-AT platform with an Arduino, we need an Arduino and a motor driver shield. We use the following two components:

- Arduino MEGA R3 Plattform
- Pololu Dual VNH5019 Motor Driver Shield

Other than the two comercial components we need a selfmade circuit board to connect the encoders. Both channels of both encoders have each to be connected to one interrupt pin. Also a power feed of +5V and GND is needed. The encoder values will be calculated on the Arduino. The connections are shown in figure 1 and this table:

- **Encoder right 1**: Arduino digital Pin 19 (Interrupt 4)
- **Encoder right 2**: Arduino digital Pin 18
- **Encoder left 1**: Arduino digital Pin 20 (Interrupt 3)
- **Encoder left 2**: Arduino digital Pin 21
- **+5v (red)**: Arduino 5V
- **GND (black)**: Arduino GND

It is sufficient to calculate speed and direction of the rotation with two pins of each encoder, because the pioneer platform uses incremental encoders. The emergency stop is connected to pin 26 and GND. Because of the Pull-UP-Resistor integrated in the Arduino the state of the emergency button can be checked by reading the value of pin 26. A pressed emergency button equals an HIGH signal - a released emergency button equals a LOW signal.

To get a secure system you should use the second circuit of the emergency button to connect a normally opened relais which cuts power when the emergency button is pressed.

B. Assembly

The motor driver shield and the encoder board are plugged on top of the Arduino. The next step is to connect the motors to the motor driver shield. Each side’s wheels are connected to two motors over a chain, such that the direction of both motors on one side is always reversed. Therefore, the positive connection of one motor has to be connected to the negative connection of the other motor and both to one terminal on the motor driver shield. All connections are visualized in figure 2.

Motor power is connected to the middle terminals.

We built our encoder board with sockets to easily plug in
the encoders and connected the emergency stop with a plug. To ensure easy maintenance we mounted the Arduino with velcro to the batterie drawer, which can be seen in figure 3.

Figure 3: pioneer 3-AT with assembly

II. ARDUINO CODE

For simple communication with the Arduino we use the rosserial package. This package enables the Arduino to subscribe and publish on ROS topics. All of the following code is located in our repository.

A. Encoder

The encoder values are calculated with external interrupts. For each pin an interrupt is defined for a rising edge. Every time an interrupt is triggered there are two possibilities:

- **the motor is spinning forward**: Then the other encoder channel has value 0. Therefore the tick counter is increased.
- **the motor is spinning backward**: Then the other encoder channel has value 1. Therefore the tick counter is decreased.

For easier understanding we use positive ticks as forward movement and negative ticks as backward movement. Because of the mirror mounting of both chains, to move forward one of the encoders has to move backward. This encoder must be reversed. This is changeable with two defines (`#define RightEncoderIsReversed` and `#define LeftEncoderIsReversed`). In the pioneer 3-AT the right encoder has to be reversed, therefore `#define LeftEncoderIsReversed` is commented out. The encoder handling is adapted from Dr. Rainer Hessmer.

B. Motor controlling

To get a dynamic motor controller the settings are integrated with parameters. These parameters can be changed in the robot_platform package. They get used to calculate the current speed of both sides.

As speed control we use a linear ramp. The ramping factor is being calculated as part of the difference between target speed and current speed. The method `adjustSpeed()` increases the speed with the ramping factor if the target speed is higher and decreases it if the target speed is lower than the current speed. It is stopped when the emergency button is pressed.

C. Main Methods of the PioneerControl.ino

- `void attachInterrupts()` attaches the interrupts
- `void initMotors()` initializes the motor shield
- `void adjustSpeed()` is the main speed controller
- `void cmd_velCallback( const geometry_msgs::Twist& msg)` callback for `/cmd_vel`
- `boolean readEmergency()` reads the state of the emergency button
- `void updateEncoder()` calculates the current speed and sends the encoder values
- `void initRos()` initializes all ros components
- `void publishMessages()` publishes the ros messages
- `void getParams()` reads the parameters
- `void setup()` initializes the Arduino components
- `void loop()` main loop of the program

D. Flashing the Arduino

The Arduino is flashable through the Arduino IDE. After installation, the library folder of our package has to be copied to the library folder of the IDE. For Ubuntu this is `/sketchbook/libraries`. Afterwards inside the IDE under Tools/Board one has to choose the Arduino mega 2560. Whenever the messages are changed the library folder has to be generated. This is shown in the ROS wiki. Arduino IDE Setup section 3.3

III. ROS CONNECTION

A. Rosserial

To communicate with the Arduino we use the rosserial package with minor changes. Rosserial manages the communication via the USB cable and makes it possible to subscribe and publish onto ROS topics. Our changes are inside the rosserial_server in the serial_node. We added some code which fetches the port of the Arduino with `libudev`. The used parameters are getting added inside the launch file `pioneer.launch` from the robot_platform package.

B. robot_platform

The robot_platform package is host to launch, config, msgs, odometry publisher and a script for simple task messages.
1) **config:** Inside the `pioneer.yaml` we have the following parameters:

For odometry and speed calculation:
- `/robot_platform/wheelDiff`: distance of the main axis between the wheels
- `/robot_platform/distancePerTick`: distance the robot moves per tick

For serial communication
- `/robot_platform/Manufacturer`: udev manufacturer
- `/robot_platform/Serial`: udev serial
- `/robot_platform/Baud`: baud rate
- `/robot_platform/FallBackDevice`: port which is getting used if udev recognition fails

To get the udev parameters right one can execute `udevadm info -a -n /dev/ttyACM0`. The port `/dev/ttyACM0` has to be replaced with the Arduino port (displayed in the Arduino IDE).

2) **scripts:** Inside the subfolder scrips is the `encoder_to_odom.py` file. This script calculates the odometry and publishes it on rostopic `/odom`. Furthermore, it publishes the tf transformation. The calculation is done with help of the parameters. It isn’t done on the Arduino because the messages are big and get the serial connection to its limit.

3) **launch:** All used nodes are getting launched with the `pioneer.launch`

IV. **FIRST STARTUP**

4) **Before the first startup:**
- flash and connect the Arduino
- connect an emergency button!!
- compile and install rosserial and robot_platform

5) **First startup:**
- make sure that the motor power is switched off and the emergency button is pressed
- connect the USB cable to the Arduino and the pc
- execute `roslaunch robot_platform pioneer.launch`
- execute `rostopic echo /Arduino/robot_state`
- push the robot platform and make sure that the encoders are counting in the right direction
- release the emergency button and check if it gets detected by the Arduino
- turn on the motor power and look for the blue led on top of the motor driver shield - it should glow steady
- execute `rostopic pub /robot_platform/cmd_vel geometry_msgs/Twist linear: x: 0.1, y: 0.0, z: 0.0, angular: x: 0.0, y: 0.0, z: 0.0'` - Take care, the robot platform should move now! If the robot platform moves slowly forward try using your emergency button and if it fails use the motor power switch to stop it.

If the robot platform did move straight forward and was stoppable by the emergency button the first startup has been successfull.

If the robot platform did move, but not straight forward a first step is to check the motor connections.

Furthermore you can check, e.g with `rviz`, if transformation and odometry are getting published correctly.

V. **FURTHER STEPS**

To get your robot platform driving autonomously you can try the mapping and navigation packages, homer_mapping, homer_navigation of our repository[6]. To get it working you will additionally need to attach a laser range finder to your robot platform.