Assignment 4

ME2984

November 3, 2015

1 Submission

Assignments will be turned in as a short (∼5 minute) video showing you have successfully completed the required steps. The video must be submitted via dropbox or a shared Google Drive Folder (please keep files size down - no need for 1080P). All code and configuration files must be submitted via GitHub, and are graded based on the time they are visible on the server. Work must be done on your own robot, and using your own code. You must also submit on Scholar. **Assignments not submitted on Scholar will not be accepted**.

2 Overview

This assignment covers implementing two critical ROS nodes for operating K.H.A.N.. The first node handles interfacing with the hardware to command motors and read encoders to integrate with the larger control framework, so the system is able to convert driving directions to actual motor velocities. The second node handles reading the IR rangefinder to produce usable sensor readings for the system to reason over. Implementing every component of this stack is a difficult, time-consuming process which would normally require completing several other courses to fully comprehend; luckily, ROS provides many packages which already implement these components. Instead of attempting to tackle every problem, you will use these packages to handle many of these challenges, which when combined with code written for previous assignments and the provided starter code, will only need to write a few new pieces of code as described below.

3 Control Interface

In order to link the KHAN hardware to the same control interface used when launching KHAN in Gazebo, the software must be updated to build the new interface with ros_control. In order to do this, the khan_robot repository must be updated:

```
cd ~/catkin_ws/src/khan_robot
git pull
```

```
cd ~/catkin_ws
catkin_make install
```

These commands will pull the latest version of the khan_robot[1] repository, then build it for use. The new node, `khan_control`, reads in a Twist[2] message containing linear and angular velocities, and performs the following tasks:

1. Uses inverse kinematics of a differential-drive[3] system to compute individual motor velocities.
2. Implements a PID control loop on velocity for each individual motor, complete with an online retunable set of parameters using dynamic_reconfigure[4].
3. Optionally computes the forward kinematics from commands to compute a dead reckoning estimate of the robot's pose in the world.

In order to use this interface, a ROS node needs to be implemented to provide the appropriate interfaces. This assignment requires creating a new ROS node, which will handle commanding motors and reading encoder data.

3.1 Motor Commands

The ROS node must respond to commanded velocities, and spin the motors at an appropriate rate. The commands for each motor are published as JointState[5] messages, with each motor having an individual topic. These commands are direct velocity commands for each individual motor, one motor command per topic, with velocities expressed in $\frac{rad}{s}$. The topics are named in a standard convention of the form `/py_controller/$motor_name/cmd`, where `$motor_name` is the name of a motor defined in the URDF (e.g. `front_left_wheel`, `rear_right_wheel`.) The node must subscribe to each of the four topics, and convert the velocity contained within to the outputs to drive the specified motor correctly. This includes both direction (as defined by the sign of the velocity) and magnitude. Note that this requires using IO commands as used with the Adafruit BBIO library in Assignment 3, adjusted to respond to received commands.

3.2 Encoder Readings

The ROS node must read the signals for encoders, and publish them for the PID controllers upstream to update appropriately. A JointState message must be published for each motor, correctly reporting the position and velocity of the motor. Positions and velocities should be reported as radians and $\frac{rad}{s}`respectively. These messages must be published on one topic per motor, following`

[1]https://github.com/FleetRobotics/khan_robot
[2]http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html
[3]https://github.com/ros-controls/ros_controllers/tree/indigo-devel/diff_drive_controller
[4]http://wiki.ros.org/dynamic_reconfigure
[5]http://docs.ros.org/indigo/api/sensor_msgs/html/msg/JointState.html

`the format`/py_controller/$motor_name/encoder, `with`$motor_name‘ matching the name of the motor in the URDF. The Quadrature Decoder written for Assignment 2 will be re-usable here for converting the encoder signals to position/velocity values.

3.3 PID Tuning

In order to get the accurate and robust motion from the motors, a PID control loop is already implemented in the K.H.A.N. hardware interface provided.the PID control parameters for each motor should be tuned. These parameters adjust how the velocity commands sent to the robot will modulate based on errors in actual execution.

1. Create a copy of `khan_control/config/example_khan.yaml` and `khan_launch/launch/example_khan_hw.launch` into `config` and `launch` directories inside the ROS package that contains the previously created ROS node.
2. Modify line 7 of the launch file to load the new config file. The key differences will be which package name is passed to the find statement, and the name of the config file.
3. Modify the config file to contain parameters to better operate the robot. This can be done in many ways, ranging from rigorously deriving desired values and tweaking, to performing a sequence of guess-and-check tests.
4. Add the ROS node created in the previous steps to the launch file, which will integrate the hardware interface with the control layer. This can be verified by using the same methods for sending driving commands to the robot as done in Assignment 1, using either rqt or rostopic.

4 Ranging Node

In order to use the IR Rangefinder, a ROS Node must be implemented to read the analog signal from the Rangefinder, convert the reading to a metric range (e.g. a value in meters, not volts or ticks), and publish the results as a Range[6] message. The Node should output on the topic `/range`.

5 Useful Tips

1. A ROS package can contain multiple ROS nodes. Remember, a ROS package is a folder inside the `catkin_ws/src` directory which contains two specific files (`CMakeLists.txt` and `package.xml`) which describe the package and what it contains. A ROS node is an application which is set up to work as part of ROS. This assignment will likely be easiest to create additional nodes inside the ROS package created for Assignment 2, since that package is already building, and contains the quadrature code.

[6]http://docs.ros.org/indigo/api/sensor_msgs/html/msg/Range.html

2. To make a Python script executable in ROS, you need to navigate to the file location (e.g. `~/catkin_ws/src/$my_repo/$my_package/scripts`) and execute `chmod +x *.py`. You should also edit the `~/catkin_ws/src/$my_repo/$my_package/CMakeLists.txt` to include your scripts in the installation command for Python scripts (which should also be uncommented so it actually executes.)

3. To check if your Python scripts are being correctly detected by ROS, you can open a new terminal and type `rosrun $my_repo`, then press tab a few times. It should start listing every node ROS detects in that package - if your node isn't listed, ROS cannot detect it for some reason.

4. Using rqt, the dynamic_reconfigure system allows online tuning of the PID parameters for each motor. This can drastically speed up testing values.

5. For ensuring that encoder signals are accurately read, consider using the `wait_for_edge()` function in the BBIO library.

6. For converting the ranges, refer to the datasheet[7] to get the details of converting the ranges. It may be useful to limit the ranges used to avoid ambiguities.

7. The URDF can be found in `khan_robot/khan_description/urdf`, with the base file being `khan.urdf.xacro`.

[7]https://www.sparkfun.com/datasheets/Sensors/Infrared/gp2y0a02yk_e.pdf