Poli V2 Robot Guide

February 16, 2018

University of Texas at Austin
Socially Intelligent Machines Lab

Electrical and Computer Engineering
University of Texas at Austin
http://www.ece.utexas.edu/

Title:
Poli V2

Project Period:
2017

Participant(s):
Prashant Rao
Alfredo Serrato
Maxwell Svetlik

Supervisor(s):
Dr. Andrea Thomaz

Page Numbers: 36

Date of Completion:
February 16, 2018

Contents

Chapter 1

Robot Overview

This chapter goes over the components of the robot and how they interact with one another at a high level. For more detailed information about a specific component and examples running a component through ROS see chapter 5.

1.1 Sensors

There are three primary types of sensors aboard PoliV2: lidar, sonar and RGB-D. Two lidar sensors, a Hokuyo 30EW at the front, and a Hokuyo 10LX at the rear, are used for primary localization and obstacle detection.

A Maxbotix 1230-EZ sonar sensor is used for additional obstacle detection of surfaces not detectable by 2D laser, e.g. glass or stairs.

Finally an Astra RGB-D camera is used for general perception. While possible to fuse depth data into localization and obstacle detection, this is not done for the Astra whose primary perception is used for manipulation of objects.

1.2 Manipulation

Robotic manipulation is accomplished with a custom Kinova Jaco2 7DOF robotic manipulator fitted with a Weiss WSG-32 gripper with force-sensing finger tips. Both the arm and gripper are exposed to the ROS network.

The robot also has two additional degrees of freedom in the pan-tilt construction on the head and neck of the robot. This is accomplished with a Dynamixel X Series servo for the pan degree, and a Maxon motor with planetary gearbox for the tilt degree.

1.3 Platform

The robot's mobility is provided via the Segway RMP 110 platform, which offers two wheeled differential drive. The acceleration and velocity capabilities are quite high and are typically scaled back via soft limits.

1.3.1 Platform Power

Power is provided in through two means: propulsion power and auxiliary power. Both are provided and charged through the RMP platform. Propulsion is strictly for internal use of the base.

The auxiliary power supplies the computers, arm, pillar, and other components. As it stands, this allows for around 4.5 hours of runtime while disconnected from the charger. If running the robot connected to the charger, expect about 12 hours of run time before the auxiliary battery is depleted.

Charging the batteries takes 8 to 10 hours due to the chemistry of the batteries. See details about charging in Section 2.1.3.

1.4 Computers

There are two computers present, each an Intel NUC *NUC6i7KYK* featuring a 2.6 GHz processor, 32 Gb of RAM, and Integrated Iris 580 graphics. Both PCs are networked locally and have access to network resources. Additionally, each PC is responsible for a number of locally connected USB devices. More on this in Chapter 3.1.

The PCs are enumerated with the robot as a base name, making them `poli1` and `poli2`. There is a defacto user account, `poli` on each computer.

1.4.1 ROS Environment

All computers run Ubuntu 16.04 LTS and ROS Kinetic.

1.5 Glossary of terms

In the following chapters, specific terms are used to refer to the robot and its components. Please refer below to help disambiguate.

- PoliV2/Poli2 - The name for an instance of the whole robot platform

- poli1 - The name for a PC on the PoliV2 platform, specifically the PC mounted on the base.
- poli2 - The name for a PC on the PoliV2 platform, specifically the PC mounted on the torso.
- poli2 high level repository - The name for the organizational repository for the PoliV2 platform. Can be found at https://github.com/si-machines/poli2

Chapter 2

Getting Started: New User with Existing Robot

This chapter will get a new user familiar enough with the PoliV2 platform that they will be able to:

- Create their own user accounts on PoliV2 computers
- Turn on the robot and auxiliary systems
- Start necessary robot services on their own user account(s)
- Network their own PC to network into the roscore of the robot

It will be helpful to have a high level understanding of the robot and its systems. Before continuing, please look over Chapter 1.

2.1 Turning on the hardware

The Segway RMP base contains two batteries- one for base propulsion and one for auxiliary systems. Both are activated and charged in parallel, making turn on and turn off procedures simple. Many components, like the router, pillar and lidar, will be powered up without direct intervention.

2.1.1 Turning on the robot

There are four main components that need to be powered manually. These are

- The Segway base
- The two PCs
- The Kinova arm

The Segway base

Locate the base control box, containing the power button and an Emergency-Stop button (Figure 2.1. Pressing the power button initiates the base's power up sequence or power down sequence, depending on the base's initial state.

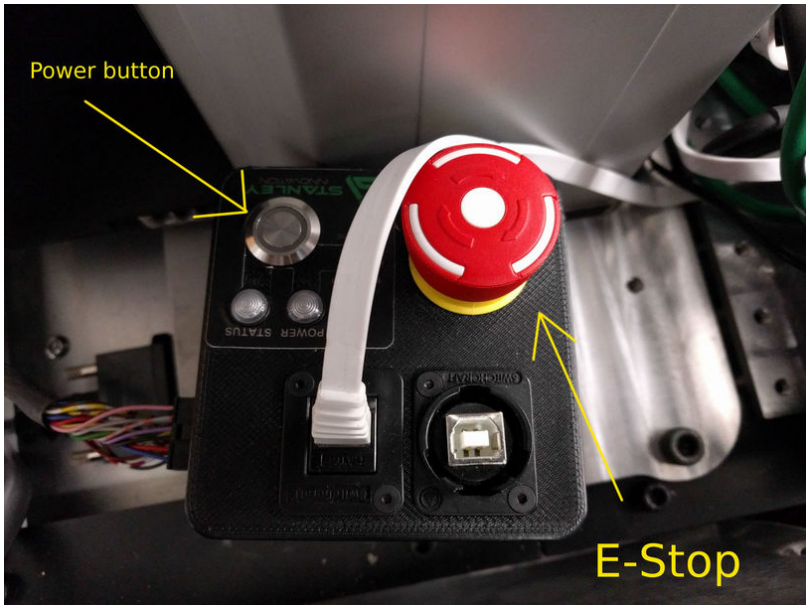

Figure 2.1: The Segway base control box

During boot up and boot down, the base emits a tone that lets you know systems are normal. You can also verify the lights on the control box.

PCs- Poli1 and Poli2

The PCs also turn on at the press of a button. Since they use power supplied by the base, the base must already be turned on before attempting to power the PCs.

See Figure 2.3 and Figure 2.4 for help finding the power buttons. The power button on each computer should light up once powered.

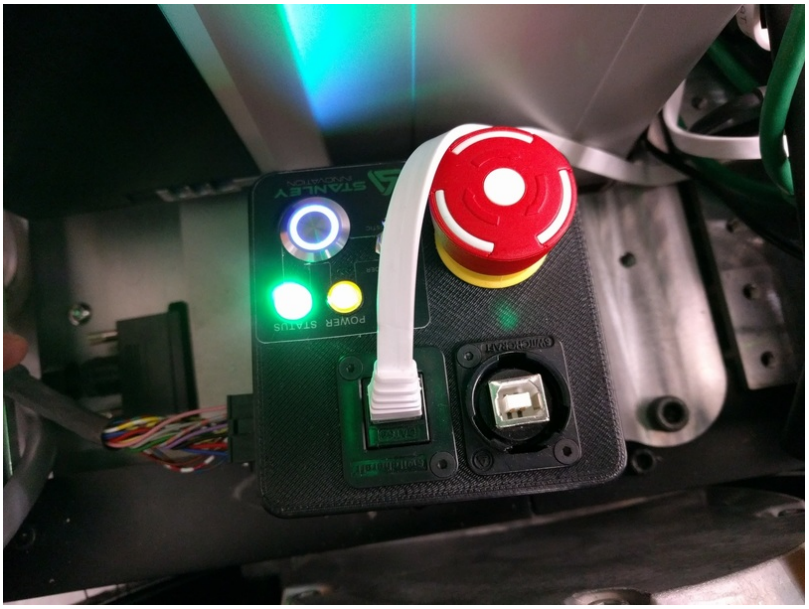

Figure 2.2: The control box power button is pulsing slowly and the indication LED is green. Both indicate system has booted normally.

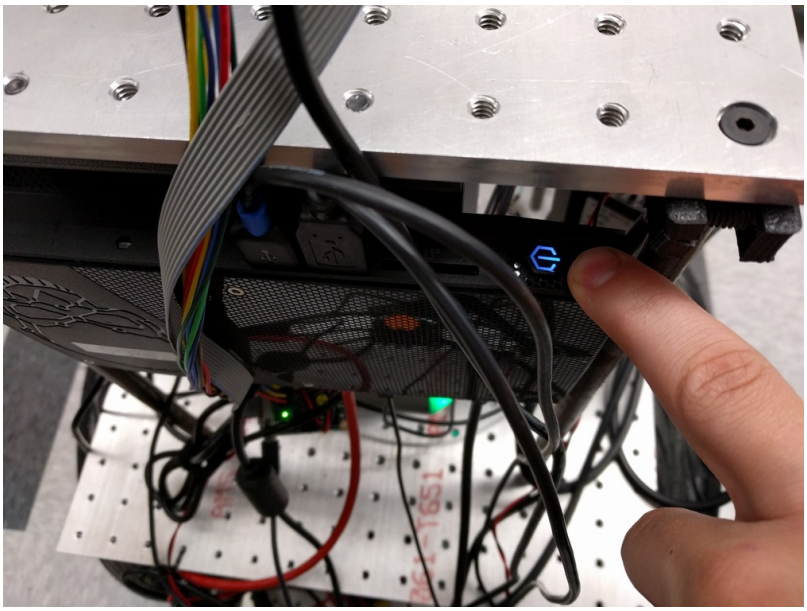

Figure 2.3: Location of the power button for PC1.

Figure 2.4: Location of the power button for PC2.

Arm

Since the arm is back-drivable when unpowered, it is sometimes advantageous to turn the arm off manually and move it to a safe position. If the previous operator has done this, you will want to check if the arm's power button is active.

Locate the arm's access panel- with power and USB inputs. The power button also exists on this panel. The arm is powered if you cannot move it or if its holding its own weight without falling.

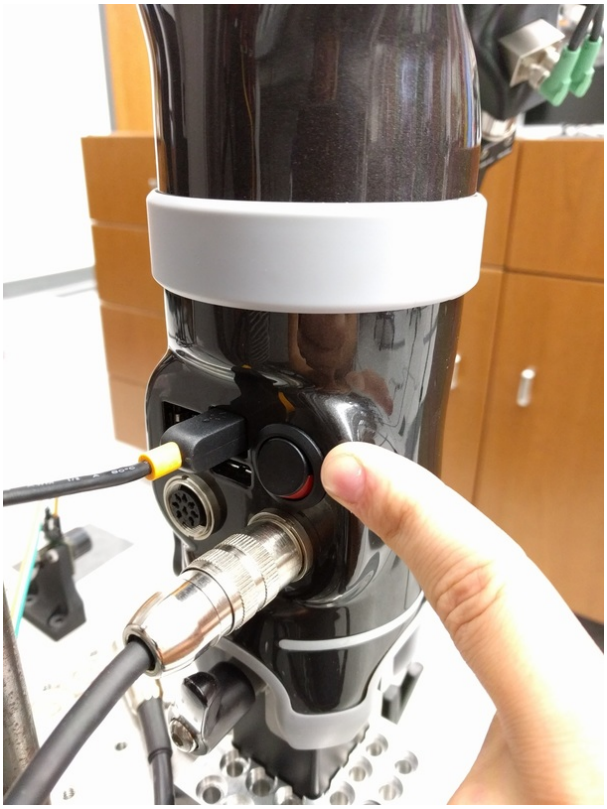

Figure 2.5: Location of the power button for the Kinova Jaco2. In this case, the arm is on.

2.1.2 Turning off the robot

This basically happens in the reverse sequence of turning the robot on.

First you should confirm the safety of the arm position and turn it off manually if necessary. See Figure 2.5 for the power button location.

Secondly, the PCs should be soft-shutdown, if possible. This means running

the `sudo shutdown now` command in a terminal in a ssh session with the PC. If necessary, you can hold down the power button to shut the PC down.

Lastly, press the base's powerbutton on its control box. An audible tone should sound, and the LED should blink red. It will be fully powered down in around a minute.

2.1.3 Charging the robot

The robot charges through a high voltage, VGA style connection directly into the base (see Figure 2.6). The cable is held into the base through two thumb-screws.

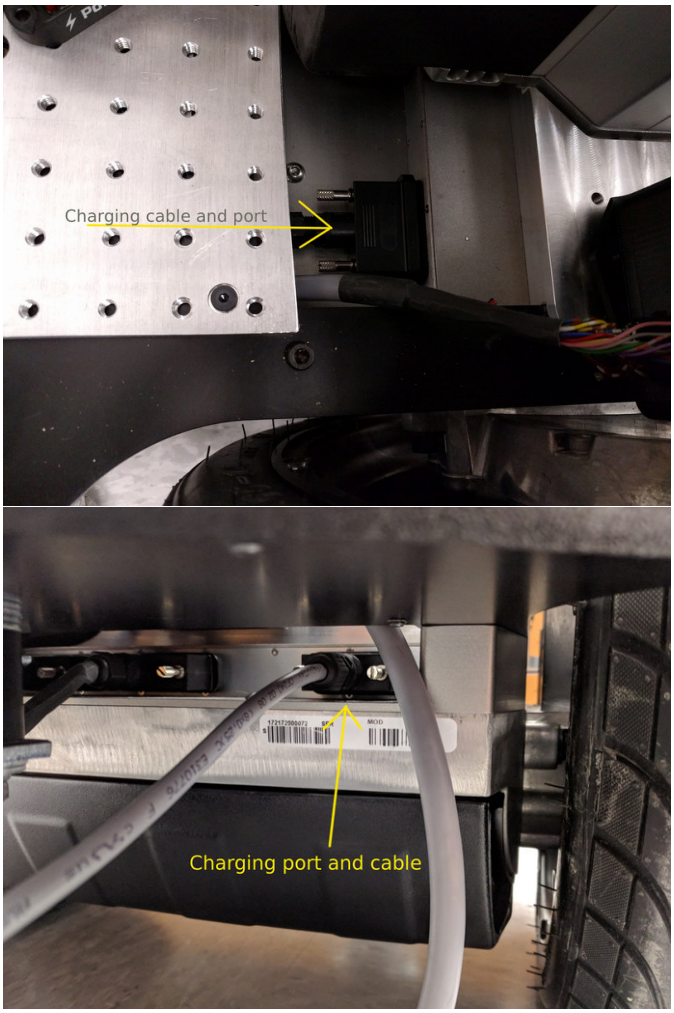

Figure 2.6: Top and bottom view of the charging cable connector for the Segway RMP.

NOTE It is important that the charger be turned **OFF** when un/plugging the charging cable to the base. It is also important than the cable be inserted evenly, and not at an angle. Either of these can cause a momentary short and can tax the charger.

Figure 2.7: Back view of the Segway base charger. The charger's power switch is indicated.

2.1.4 Disconnecting robot from charger

Turn the charger off using its power switch on its back panel. Unthread the thumb screws from the connector and the base. Pull the charging cable backwards, off the connector, with even force.

See Figure 2.6 and Figure 2.7 for details.

2.2 Create user accounts

2.2.1 Setting up local hosts file

Examples in this section make use of a hostname alias, rather than specifying the IP address of **poli1**. To add **poli1** and **poli2** as hostnames, see Example 2.1. This should already be done on the robot's computers but will need to be done for any computer used to access the robot's computers or network.

Example 2.1 (Example of setting hosts file for robot use)
On your local computer (assuming Ubuntu)

- Open `/etc/hosts` with sudo privileges

- Add `10.66.171.91 poli1`
- Add `10.66.171.24 poli2`
- Save and exit

IP addresses are assumed to be set according to Section 3.1

2.2.2 Initial Account Setup

Connect to the robot's wireless network, which should be of the form `PoliV2-N`, where N is a number designating the enumeration of that particular robot. It is brought up when the router boots, which takes a couple minutes after turning on the robot. Since the robot needs to be powered for this, go ahead and look through Section 2.1 to turn the robot on.

You can connect to the network using the password **simachines**. This same password is used for access to the default accounts on **poli1** and **poli2** computers.

We will now go through the process of adding a user account with the correct permissions to one of the PCs, **poli1**.

Example 2.2 (Example of setting up user account on poli1)

- ssh into poli1: `ssh poli@poli1`
- Make yourself a user: `sudo adduser <your desired username>`
- Add user to groups: `sudo usermod -aG sudo,dialout,cdrom,adm,dip,plugdev,lpadmin,sambashare,audio <your username>`
- Log off: `exit`
- Set up your ssh keys for your new user: `ssh-copy-id <your user>@poli1`
- Log in to your new user account: `ssh <your user>@poli1`

Now, log into your account and create a catkin workspace in your home directory. You can see how to do that at
http://wiki.ros.org/catkin/Tutorials/create_a_workspace.

If successful, do the same processes for **poli2**. Don't forget to setup your bash environment as in Example 2.3, 2.4.

Example 2.3 (Setting up initial .bashrc file: POLI1)
Open `~/.bashrc`
At the bottom add `source ~/catkin_ws/devel/setup.bash`
Save and close
In a terminal enter the command: `source ~/.bashrc`

Example 2.4 (Setting up initial .bashrc file: POLI2)
Open `~/.bashrc`
At the bottom add `source ~/catkin_ws/devel/setup.bash`
At the bottom add `source poli_RMP.bash`
At the bottom add `export ROS_MASTER_URI=http://poli1:11311`
Save and close
In a terminal enter the command: `source ~/.bashrc`

Finally, you need to setup wireless internet under your user on **poli1** and **poli2**. This requires logging in by connecting a keyboard, mouse and monitor directly to the robot and setting up a connection to a network through Ubuntu's `network manager`.

There exist specific WiFi accounts for the robots to connect to the `utexas` network. Contact a lab manager for these credentials.

If `network manager`'s front face is updated to be X-forwarded or you switch to a different manager, you may be able to set up the wifi connection in a terminal via `ssh`.

2.3 Setting up the code base

Note: This section of the guide is possibly outdated. Please see the software wiki for the latest.

There are a fair number of forks and specific branches that need to be checked out to get all systems working. Luckily the ROS-install system makes fetching these simple.

2.3.1 Fetch high level repository

Example 2.5 (Setting up the PoliV2 codebase)
Assuming catkin_ws is initialized and using a PoliV2 computer.

- `git clone https://github.com/si-machines/poli2.git` in `src` of catkin_ws
- cd into root of `catkin workspace`
- `rosdep update`
- `rosdep install --from-paths src --ignore-src --rosdistro=kinetic -y`
- `cd ~/catkin_ws/src`
- `wstool init ./ ./poli2/dependencies.rosinstall`
- `sudo apt-get purge ros-kinetic-dynamixel-workbench-toolbox`
- `cd ~/catkin_ws`
- `catkin_make`

In the future you can update from all repos using the `wstool update` command.

We use `rosdep` to fetch packages on the public repository list (reading from `package.xml`) and `wstool` to fetch specific branches of specific forks as specified in the `.rosinstall` file.

Note that the codebase should be set up on both **poli1** and **poli2**, so repeat Example 2.5 on both PCs.

Important note: The last thing needed is to copy the `poli_RMP.bash` file to the root of your home directory. This file is required by the `segway_ros` driver.
As long as Examples 2.6 and 2.4 have been followed, the RMP environment variables should be set in perpetuity.

Example 2.6 (Setup RMP Environment Variables)
- `roscd poli_launch`
- `cp config ~/poli_RMP.bash`
- `source ~/.bashrc`

2.4 Starting ROS and critical systems

Since PoliV2 is a multi-purpose research platform, and will be used by many separate users, there are no assumptions made about the virtual environment. As such, no ***device*** launch files are brought up on startup or log-in; though the ability to do this is fully supported. See 'Automating Bringup Procedure' Section on the Poli2 Wiki.

However, a single launch file is brought up on startup to start the **ros core**. This is to prevent accidental preemption of the core, and to allow some (typically) persistent parameters to remain on the roscore session, e.g. robot_description.

Thus, with the robot's systems on and the codebase installed and compiled, we're ready to bring up the systems manually.

2.4.1 Poli1 PC

As detailed in Section 3.1, each PC can only bring up the devices it has connected to it, plus any networked components.

We can bring up all poli1's devices as in Example 2.7.

Example 2.7 (Bringup poli1's devices)

```
ssh user@poli1
roslaunch poli_launch poli1_bringup.launch
```

2.4.2 Poli2 PC

We can bring up all poli2's devices as in Example 2.8. `poli2_bringup.launch` does the heavy lifting, bringing up the navigation stack, MoveIt!, and all networked components.

Example 2.8 (Bringup poli2's devices)

```
ssh user@poli2
roslaunch poli_launch poli2_bringup.launch
```
Wait several seconds
```
roslaunch poli_launch scan_merger.launch
```

Note that you must launch the scan_merger.launch file after bringing up everything else on **poli2**. This is due to a bug in the implementation of the scan merger

node. However, it is still necessary to run, due to the navigation stack's reliance on the merged tf frame that the scan merger provides. Otherwise, issues with the tf tree ensue.

Now all the robot's systems should be up. If you run into problems, check the troubleshooting page on the poli2 high level repository or in Chapter 7 of this document. You can see specific examples of how to move components or access robot state in Chapter 5.

2.4.3 Networking the Robot With Your Computer

This allows you to run graphical elements (RViz, image_view, etc) on your local machine that may be difficult or impossible to view on the robot's computers directly.

Example 2.9 (Reserving IP on the robot's network)

- On your PC, connect to the desired robot's WiFi
- Go to the router's webpage: `10.66.171.1`
- Login
- Click **Advanced**
- Under **Network** in the navigation pane, select **DHCP server**
- There will be a table of MAC addresses and IP addresses...
- Find the **name** entry of your host and click **bind**.
- Copy the IP address that your machine is bound to.

Example 2.10 (Setting up ROS Environment)

- Edit your `~/.bashrc` file
- At the bottom add `export ROS_IP=xyz` where `xyz` is the IP your machine is bound to on the network.
- At the bottom add `export ROS_MASTER_URI=http://poli1:11311`
- Save and close

- Source your .bashrc: `source ~/.bashrc`

In the event you want to run a roscore locally, simply change this file again, but where `ROS_IP=127.0.0.1` and `ROS_MASTER_URI=http://localhost:11311`.

Example 2.9 walks through binding your IP, which prevents getting kicked off the ROS network when your IP lease expires. Example 2.10 walks through setting up the environment variables needed to network through the robot.

Note that this would have to be done on each robot you want to work with, since each has its own network. Therefore it will be easier to reserve the same IP address on each robot network, preventing you from needing to update your `~/.bashrc` with each new robot.

You can read more about ROS networking at http://wiki.ros.org/ROS/NetworkSetup.

Chapter 3

Robot Organization

3.1 Network Configuration

Each robot has its own local area network (LAN) that facilitates connectivity of components and computation.

Not all components are hardwired to the network; some must be connected to a PC via USB. As such, the components connected directly to a PC are local to that PC and can only be brought up on that PC. See Figure 3.1 for details on the connectivity of the system.

3.2 Codebase

At the time of this writing, the codebase depends heavily on forks and branches of existing opensource packages and libraries. This section briefly looks at which fork/branch combination is used and why it was made.

Note that fork/branch resolution happens automatically through `wstool` and that this section is for informational purposes only.

- segway_v3 - Stanley Innovation - `master` : no kinetic release
- wsg_32_description - SI-Machines - `master` : no kinetic release
- ira_laser_tools - iralabdisco - `master` : no kinetic release
- kinova_ros - GT-RAIL - `7dof_kinetic` : no kinetic release + 7dof changes. Follows `7dof` branch of same fork.
- wsg50-ros-pkg - SI-Machines - `kinetic-devel` : no kinetic release; URDF and driver bug fixes.

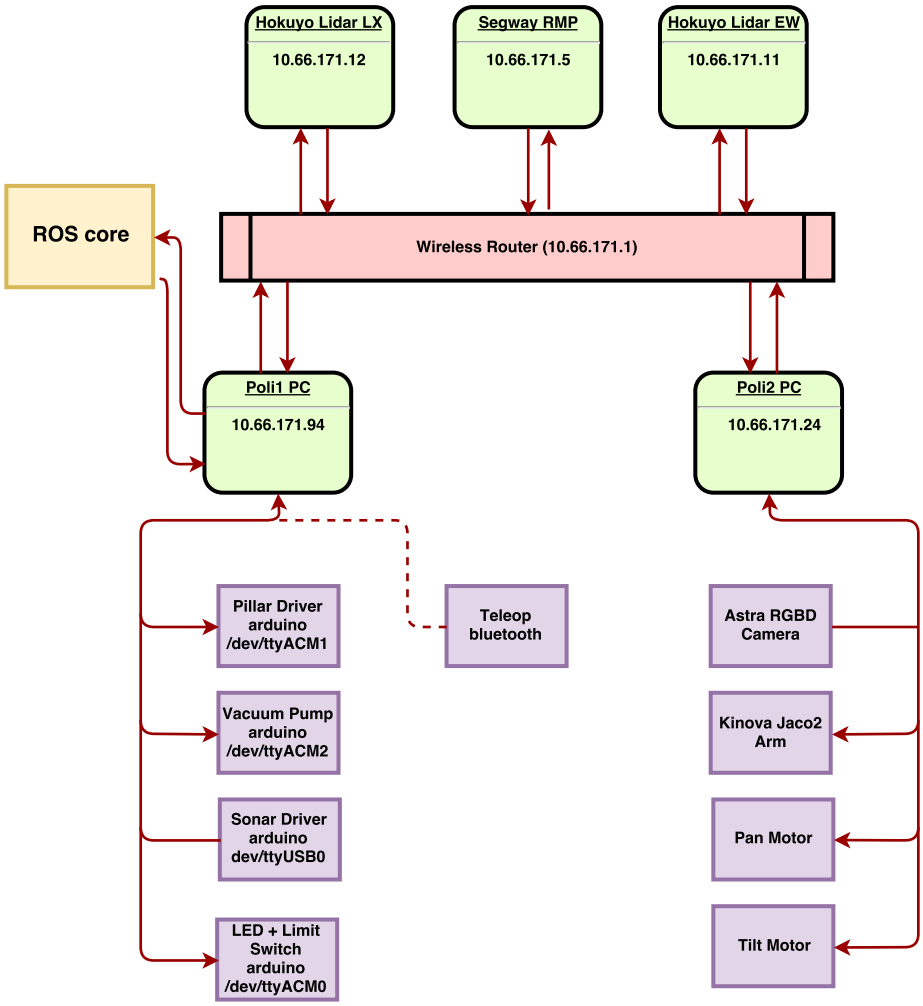

Figure 3.1: Network map for the PoliV2 Robot Platform

- epos_hardware - RIVeR-Labs - `kinetic-devel` : no kinetic release
- dynamixel-workbench - ROBOTIS-GIT - `master` : Kinetic release is broken
- dynamixel-workbench-msgs - ROBOTIS-GIT - `master` : Kinetic release is broken

3.2.1 Microcontroller development

All microcontroller components are arduino IDE compatible. All files used to flash onto the particular arduino can be found in `scripts/arduino/` folder of the `poli_launch` package.

A guide on how to do development and flash to Arduinos is available at the github wiki: https://github.com/si-machines/poli2/wiki/Arduino-Development.

Note that the Arduino devices names are given in Figure 3.1. These device names are definite for the first Poli2 robot, but may change on subsequent robots depending on how device names are assigned.

3.2.2 Microcontroller wiring diagrams

LED + limit switch

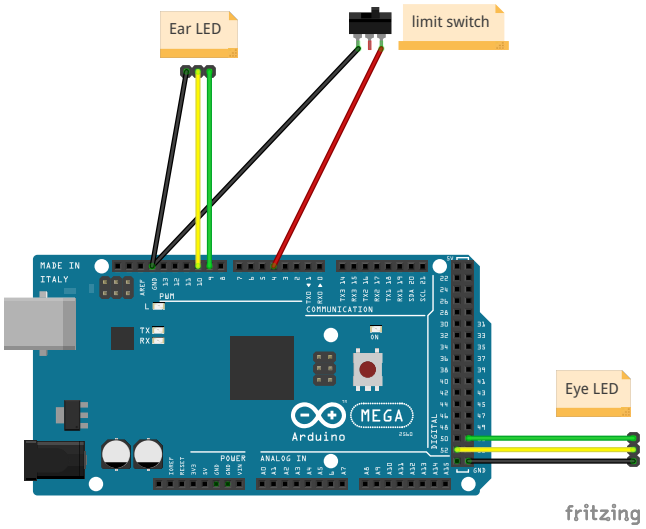

Figure 3.2: Wiring diagram for the LED and limit switch arduino

Pillar

Vacuum

Chapter 4

Setting up a New Robot

This section contains all information pertaining to setting up and configuring the components of the robot as they come from the factory. These changes are strictly software configuration necessary for the components to communicate and would only need to be carried out once per robot.

4.1 Setting up Network Components

The Segway RMP base is set in the 10.66.171.x sub net. This requires us to change all other networked devices (router, lasers, computers) to accept being addressed in this range. For simplicity, we try to keep all IP addresses consistent across all robots. Reference Figure 3.1 when setting the IPs of the various components.

4.1.1 Setting up network hosts

In order to resolve network names and communicate over a common network, both **poli1** and **poli2** need each others names resolved. See example 4.1

Example 4.1 (Configure network hosts)

- On **poli1**
- Edit `/etc/hosts` with `sudo`
- Add `10.66.171.24 poli2`
- Save and exit
- On **poli2**
- Edit `/etc/hosts` with `sudo`
- Add `10.66.171.91 poli1`

- Save and exit

4.1.2 Setting up the connections in Ubuntu

Example 4.2 is required to simultaneously use Internet via WiFi and LAN via ethernet. This manually manages the ethernet connection that allows connection to the LAN and prevents Ubuntu from using it as an outbound route.

Example 4.2 (Configuring Ubuntu's network manager)

```
 auto lo
iface lo inet loopback
auto eno1
iface eno1 inet static
address 10.66.171.x
netmask 255.255.255.0
```

- Edit `etc/network/interfaces` file with `sudo`
- Fill in the file with the above codeblock
- Change the IP address after `address` to match the IP of the computer whose file you're editing.
- Restart computer.

4.1.3 Setting up network connections for the router

When setting up the main router, change the router's IP address to `10.66.171.1`.

While you're there, change the network name to something appropriate (SSID = PoliV2-N) where N == a number that won't cause collisions with existing robot networks.

Finally, go to the DHCP client table. Here you will be able to bind IP addresses of important components. You will need to bind the IPs of the wireless bridge, `poli1` and `poli2` at a minimum.

4.1.4 Change the IP Addresses of the lasers

The default IP address of the Hokukyo lasers are 192.168.0.10

You can find a link to the Windows IP changing tool provided by Hokuyo in Section 6.1.1. The ROS `urg_node`, which is what we use to fetch data from the lasers, also has a node called `change_ip_address`. This has not been tested by our group, but may work in place of the Windows tool.

Change the laser IPs to the appropriate value. Ensure the correct laser gets the expected IP address.

4.1.5 Configure the Gripper's IP

Next, configure the Gripper's IP to be statically 10.66.171.21. This can be accomplished through the gripper's web interface at its default IP address, `192.168.1.20`.

This should be done outside the immediate network, though, if you've already changed the IP range of the router.

4.1.6 Configure the Gripper's Wireless Bridge

Finally, configure the wireless bridge to be a client of the router and reset its IP range. The bridge has a default IP address of `192.168.1.1`.

Example 4.3 (Reconfiguring the wireless bridge)

- Connect to bridge's LAN interface
- Navigate to `Quick Setup` on the left navigation panel
- Click Next
- Select **Client** under 'At Home'
- Click Next
- Select the local router SSID for the robot you're configuring
- Enter the correct security credentials
- Finish. Results in a reboot.
- Select **Network** on the left navigation panel
- Select **Static IP** for type
- Enter **10.66.171.2** for IP address
- Enter **10.66.171.1** for the gateway

Chapter 5

Component Specifics

This chapter details the particulars of each component. This includes how to interface with the component through the high level repository Poli2 and its dependent packages and libraries. Unless specifically specified, units are in meters, kilograms and radians.

5.1 Sensors

5.1.1 Lidar

The lidar are used for both localization and obstacle detection. In order to use multiple scan sources for localization using existing algorithms, they are merged into a single scan using `ira_laser_tools`. This package has a bug in implementation that requires the merging node be launched after the lidar drivers are brought up.

5.1.2 Sonar

The sonar are used strictly for obstacle detection and have a range of approximately 5 feet at their current mounting points. As it stands, the sonar are better used for clear and reflective surfaces that the lidar cannot pickup. Cliff detection is not possible with the current configuration of sonars.

5.2 Manipulation

Arm-based manipulation has two hardware components due to the use of a third party gripper (WSG-32) with the manipulator (Jaco2 7DOF). These components are exposed to ROS separately. We also consider the pan-tilt motor system which, though have different controllers, are exposed to ROS under the same interface ros-control.

5.2.1 Jaco2 7DOF

The Jaco2 uses the same ROS driver that previous Kinova products use. So most things (services, actions, messages) should be familiar. Do note that we use the `kinetic_7dof` branch of the GT-RAIL fork of `kinova_ros`, and that any finger services in the API will not work with the third party gripper.

Arm Driver

The majority of the driver functions are identical for the 7dof Jaco. The most obvious change is the name of the arm as shown as a prefix for all driver topics and services.

The arm has been configured for gravity compensation to be used in an upright position with the third party gripper. Changing either of these parameters would necessitate changing the driver's gravity vector and regenerating center of mass parameters.

> **Example 5.1 (Start gravity compensation)**
> `rosservice call /j2s7s300_driver/in/start_force_control "{}"`
> Note that the 'start_grav_comp' service doesn't seem to do anything.

Arm MoveIt! Configurations

There exist two separate MoveIt! configurations for the custom Jaco2 arm: `jaco2_custom_arm_moveit_config` and `poli2_full_config`.

The former is a standalone configuration for the arm and the Weiss gripper, useful when running manipulation on a table top. The latter is a configuration which takes into account the entire PoliV2 robot platform and is thus collision-aware of the robot's components.

The `poli2_full_config` is run by default in the startup launch files. `jaco2_custom_arm_moveit_config` is kept only as a useful addition.

5.2.2 Vacuum Gripping Unit

There is a ROS-controllable vacuum unit that would allow high level vacuum manipulation plans. This unit is controlled by an arduino-based microcontroller and is connected to **poli1** via USB.

Since this unit may or may not always be installed to the robot, ensure that the box is plugged into a **12V** power rail.
See Example 5.2 for how to command the gripper system.

There are three commands that the interface accepts.

- Start pump: begins pulling a vacuum
- Stop pump: stops pulling a vacuum
- Release object: Requires the pump be running. Briefly stops pulling vacuum long enough to release an object that gripper may be holding.

> **Example 5.2 (Start vacuum unit)**
> `rosservice call /poli/gripper_vacuum "command:  0"`
> Find the mappings of commands in the GripperPump.srv service description file in `poli_msgs`.

5.2.3 Robotiq Gripper

TODO

5.2.4 Weiss WSG-32 Gripper

NOTE the Weiss gripper has been swapped for a Robotiq gripper, so the following does not apply. This information is retained in the event that Weiss is swapped back or used independently of the robot.

The Weiss gripper has a number of services and a single control topic. Here are several examples of commanding the gripper through ROS.

The gripper is wirelessly interfaced through the network and is accessible through a web interface. Navigating to the address `10.66.171.21`, the Gripper's IP address, should bring up the gripper's landing page. This page allows manual control of position, debugging and settings management.

Occasionally the gripper may run into a fault that must be manually cleared. You can clear the fault through the web interface or as in Example **??**.

> **Example 5.3 (Send position goal to Gripper)**
> `rostopic pub /wsg_50_driver/wsg/goal_position std_msgs/Float64 "data: 0.0"`

The mode can be left blank, but velocity must be non-zero.
Units are in mm.
Max range is 65 mm. Min range is 0mm.

5.2.5 Pan-tilt

Tilt system

The tilt motor is a high torque Maxon motor with an EPOS motor controller. This motor and control comes with an important caveat: relative encoders. This means **the tilt motor must be in the full down position when the motor first gets power**.

With this in mind, there is a software check that happens on the first launch of the bringup files on a given `roscore`. This software checks the following conditions:

- The tilt driver is reporting a value of 0.0
- The limit switch inside the head is pressed

where the limit switch is positioned in the 0.0 position expected by the URDF model.

If both conditions are met, a flag is set as a `rosparam` and the tilt controller is brought up. Otherwise, the tilt motor's encoders have been incorrectly initialized, so the user is notified and the tilt controller is not brought up.

Pan-tilt control

The pan and tilt motors are controlled through `ros_controllers` position controllers. Thus they are both controlled through ROS in the same way.

Units are in radians, and limits are specified in the URDF for that particular joint.

Example 5.4 (Send position goal to pan motor)

```
rostopic pub /pan_motor/position_controller/command std_msgs/Float64
"data:  0.0"
```

Example 5.5 (Send position goal to tilt motor)
```
rostopic pub /tilt_motor/position_controller/command std_msgs/Float64
"data:  0.0"
```

5.2.6 Pillar

The elevating pillar is controlled through a topic published by a microcontroller. It has an approximate range of

$$0, 0.4$$

meters.

Example 5.6 (Send position goal to Gripper)
```
rostopic pub /pillar/command std_msgs/Float32 "data:  0.1"
```

5.2.7 LEDs

There are two LED systems in the Poli2 platform: an LED strip for the ears, and an LED matrix array for the eyes. These are both controlled by a single microcontroller in the head.

This microcontroller hosts two ros services: /led_eye and /led_ear that each use a lightweight service message to change LED state.

It is not possible to send entire arrays of bytes to control LEDs due to buffer limits in `rosserial`.

For usage examples consult the github wiki.

5.3 Platform

The base platform has one lower level method of control: velocity. You can publish velocity commands directly to the base through `rostopic` (see Example 5.7).

Generally, the base will be controlled through one of two means: teleop and/or autonomous navigation. Navigation is handled via `movebase` and can be activated via 2D navigation goals. See more on this at http://wiki.ros.org/move_base.

Example 5.7 (Publish velocity to base)
```
rostopic pub /poli/teleop/cmd_vel geometry_msgs/Twist MSG
```

Example 5.8 (Starting base teleop with playstation controller)
Pre: Bringup procedures on both PCs have been completed according to Ex 2.7 and Ex 2.8.
```
ssh poli1
roslaunch poli_navigation_apps playstation_teleop.launch
```

5.4 Computers

Each computer has a defacto account named `poli`, which can be used in a pinch. However, proper procedure is to create and use your own user account on each computer, as detailed in Section 2.2.

It should be assumed that poli1 will automatically bring up the `roscore` under the `poli` user account, which has sudo privileges. This prevents the ROS network from being killed accidentally, but means that ROS parameters can persist beyond a node's session which is problematic in certain situations.

Chapter 6

Data Sheets

Many components are off the shelf and have their own specific documentation.

Here we provide links to the item's technical data sheets and third party repositories if applicable.

6.1 Data Sheets

6.1.1 Sensors

Hokuyo 30EW

Specifications
Overview Communication Protocol
Code repository
Windows IP changer

Hokuyo 10LX

All documentation

Maxbotix 1230EZ

Specfications
Quick-start Guide

Orbecc Astra

Specifications
Code repository

6.1.2 Manipulation

Jaco2 7DOF

Specifications
User guide
Advanced guide
Code repository

Weiss WSG-32 Gripper

Manual
Command set
Code repository
Description repository

6.1.3 Base Platform

RMP 110

Code repository

Chapter 7

Troubleshooting

7.1 Acceptable errors

7.1.1 Base

Hector Pose Estimation

```
Skipping XML Document /opt/ros/kinetic/share/
hector_pose_estimation/hector_pose_estimation_nodelets.xml which had no
Root Element.  This likely means the XML is malformed or missing.
```

This will show when launching the base. We don't use hector pose estimation, so this does not affect anything.

7.1.2 Arduino

Lost connection

```
Lost sync with device, restarting...
```

This shows up periodically, for a yet-to-be-determined reason. It does not seem to negatively affect input/output from the arduino(s).

7.2 Connectivity and Hardware Issues

7.2.1 Dynamixel driver / pan_controller driver crashes

Previously this was USB devices and udev rule issue. This now looks to be an issue with the controller driver itself.
There is a github issue open: https://github.com/si-machines/poli2/issues/40

7.2.2 TF tree broken / no laser scans output in RViz / localization fails

Exact error: 'No laser scan received (and thus no pose updates have been published) '

Whats wrong: The scan merger node was launched before the lidar drivers were fully up, or it wasn't launched at all. Running `rosrun poli_launch scan_merger.launch` in a separate terminal solves this.

7.3 Issues with existing code bases

More specific issues that arise coming from other packages or libraries should be opened as issues on the respective issue tracker.

Issues or additions to the PoliV2 repository should be opened as a ticket at https://github.com/si-machines/poli2/issues or as a Pull Request.