Fetch Robotics

Release Indigo

Fetch Robotics Inc

October 28, 2016

CONTENTS

This manual can also be downloaded as a PDF.

CHAPTER

ONE

INTRODUCTION

This manual is intended to help users successfully install, use, and develop code on Fetch and Freight Research Edition robots. The software installed on both robots is based on ROS. Please visit ros.org to learn more about ROS.

All users should read and become familiar with the safe operating procedures set out in Safety Section before operating a robot.

If you are setting up Fetch or Freight for the first time please see the tutorial videos for unboxing Fetch and Freight to learn how to properly unpackage and setup the robots for the first time.

1.1 Before you Start

Before getting started, below is an overview of what you need to use and operate Fetch and Freight safely.

- **Fetch and Freight Safety**
- Read Safety Section in its entirety before using either robot.
- **Safe Environment**
- Only operate Fetch and Freight in an environment free of hazards. Specifically, stairways and large drop offs can cause personal injury and extreme damage. Avoid dangerous objects, sharp objects (such as knives), fire sources, or hazardous chemicals. A robot with a knife or hot object is inherently dangerous.
- **Space**
- Make sure that there is enough space for Fetch and Freight to drive around and perform tasks. Both Fetch and Freight are designed to operate in ADA-compliant environments (Americans with Disabilities Act).
- **Development Tools**
- To connect with Fetch and Freight, a laptop or desktop computer is needed. Ideally Fetch and Freight can connect to a wireless network but users can also directly to Fetch or Freight via the Ethernet port on each robot.
- **Linux**
- Familiarity with the Linux command-line is strongly recommended. The computers in both Fetch and Freight have Ubuntu Linux installed. Tasks can be performed by logging in remotely by using ssh or other similar tool. Users can also directly plug in a display, mouse, and keyboard to the access panels on Fetch and Freight.
- **Electrical**
- Fetch and Freight charge using a power brick that uses a standard 120VAC or 220VAC power outlet.
- **ROS and Fetch and Freight**
- Fetch and Freight R&D software is based on ROS. Completing the beginner tutorials will help users understand how to operate and write applications for the Fetch and Freight.

- **Fetch and Freight Support**
- Please visit the Fetch Robotics support portal at http://support.fetchrobotics.com and login to review service information, modularity specifications, important safety updates, and submit hardware or software support tickets for your robot.

CHAPTER

TWO

SAFETY

Safety is extremely important to Fetch Robotics. Safe operation of robots is important yet challenging and it is important to remember that safety is a continual process that is shared by the robot designer, operator, and administrator. The following section provides an overview of the issues, safety-related design features, and a basic set of guidelines to support safety when using the Fetch and Freight R&D robots.

2.1 Safety Overview

When operating Fetch and Freight R&D robots users should always be conscious of safety. Remember the robots are heavy pieces of equipment and have moving parts. As the robots travel through an environment they can carry and manipulate a wide variety of objects. Since the Fetch & Freight R&D robots are for applications development, their moves and actions may not be entirely predictable. Both the Fetch and Freight robots can cause significant damage if they fall on or run into a person. There are also several ways that the robots can pinch, grab, or twist fingers or other body parts (these regions are labeled). Fetch can also manipulate dangerous objects and knock over heavy objects. People should always be cautious and attentive around Fetch and Freight R&D robots.

2.2 Design Features

While retaining the power of a R&D platform, both the hardware and software of Fetch and Freight are designed to minimize risk. The exterior of both Fetch and Freight clearly mark regions that could pinch or injure while mechanism is in motion or being moved by hand. Both Fetch and Freight have emergency stop buttons in case there is a need to immediately stop the motion of the robot.

In software, low-level safety limits have been incorporated to limit motor current, motor velocity, range of joint motion, and trajectory deviations. High-level applications also integrate the various on-board sensors to avoid obstacles when navigating or moving the arm.

These design features help make Fetch and Freight more robust. However a R&D robot is **never** absolutely safe. The application developers' safety, as well as the safety of others, depends on the developers' constant attention. It is important for the user to be aware of potential dangers and learn to anticipate and prevent problems.

2.3 General Usage Guidelines

While many guidelines for safe use of a robot stem from common sense, a basic set is listed below. It is important to follow these guidelines, but please note that these guidelines alone do not guarantee safety, only reduce risk.

- **Before operating** or working with a Fetch or Freight each user must:
- Watch the safety video.

- Read this user manual, specifically the entirety of Section 2 on Safety.
- **Supervise children, visitors**, and anyone who has not followed the previous guideline. In particular, make sure they:
- Do not come in range of Fetch or Freight R&D robots when active.
- Are aware that the robot could move unexpectedly and is potentially dangerous.
- Are not alone with Fetch or Freight.
- Do not operate Fetch or Freight.
- **Maintain a safe environment**. Safety is not only impacted by how a developer operates a robot, but the environment as well. The Fetch and Freight R&D robots are designed to operate in laboratory environments.
- Keep the robots **at least 5 meters from the top of a stairway or any other drop off**.
- Make sure the robots have adequate and level space for any expected or unexpected operation.
- If Fetch travels on a ramp, make sure that the spine is lowered and the arm is tucked so that the center of gravity is as low as possible. The slope of the ramp should not exceed 1:12. Also make sure that the robot cannot drive off the edge of the ramp.
- Make sure the environment is free of objects that could pose a risk if knocked, hit, or otherwise affected by the robots.
- Make sure that there are no ropes or cables that could get caught in the covers, wheels, or arm.
- Make sure that no animals are near the robots.
- Keep fingers, hair, and clothing away from wheels, gears, and any location marked as a potential pinch point.
- Be aware of the location of emergency exits and ensure that the robots cannot block them.
- Do not operate the robots outdoors.
- **The Fetch and Freight covers are flame-retardant**. However keep the robots away from open flames. Never use the robots around stoves or ovens.
- Do not allow the robot to come in contact with liquids (spilled drink, rain, etc.) If the robots do get wet, turn off the breaker switch at the back of the robot and contact Fetch Robotics.
- Before removing any covers, the robot should be unplugged and the breaker switch at the back of the robot should be off.
- Make sure that the power cord is in good condition. Cord insulation must be intact with no crack or deterioration. Both connectors should be undamaged. If the power supply is damaged in anyway, it should be discarded and replaced with a new one from Fetch Robotics.
- Do not run the robot without its covers, the covers help to protect users from internal mechanism pinch points and potential electrical shock.
- Use **common sense** when operating the Fetch and Freight R&D robots.
- Do not allow the robots to grab or hit any person.
- Do not allow the robots to drive into contact with, or over, any body part.
- Do not allow the robot to interact with any sharp or dangerous items.
- Do not allow the robot to operate potentially dangerous appliances (like stoves) or power tools.
- Pay attention to the **warning labels** on the robots.

Warning: Do not modify or remove any part of the software safety features

2.4 Warning Labels

Below are pictures of all the warning labels that can be found on the robot and associated safety issue.

2.4.1 Pinch Point

There are several pinch point warning labels on the robot. The labels mark the regions of the robot that could cause injury to hands or finger while moving. It is important to hit the run stop immediately if a finger or hand becomes trapped in a pinch point.

2.4.2 Electrical Shock

The electrical shock labels mark regions of the robot that could cause electrical shock if damaged or wet. If the water enters the battery compartment of the robot or the power intlet connector, do not continue operating the robot. Shut the robot down and turn of the robot using the power disconnect switch on the back of the robot. Then contact Fetch Robotics support.

2.4.3 Laser Beam

The laser beam warning label is to remind the user that there is an active laser scanner in the robot. The laser scanner is a class 1 laser scanner and is eye safe under all **normal** operating conditions. However it is important to note that incorrect use can lead to the user being exposed to dangerous radiation. If the laser housing is damaged on the robot do not continue using the robot or look directly into the laser beam region.

2.4.4 Read The Manual

Read the manual stickers are found beneath the skins of the robot. It is important for the user to read the manual and other maintenance documents before attempting to repair or perform maintenance on the robot.

CHAPTER

THREE

GETTING STARTED

3.1 What's In Box

The Fetch and Freight Research Edition robots each ship in reusable ATA cases. Inside each case you will find the robot, toolkit, and power supply for charging the robot.

3.1.1 Fetch and Freight

Please watch the video below for unpacking Fetch or Freight. The video covers unpacking the robot, connecting the batteries, turning on the robot, and driving the robot via the provided joystick.

3.1.2 Toolkit

The toolkit contains the tools, accessories, and fasteners needed to use the Fetch and Freight. The picture below shows packaged toolkit.

Inside the toolkit box you will find:

Item #	Item Name	QTY	Purpose
1	Metric Hex Key Set	1	For removing screws and attaching accessories
2	Right Angle USB Connector	1	To prevent USB connector damage while driving
3	Wireless Joystick	1	For teleoperating the robot
4	USB Cable	1	For charging the wireless joystick
5	Finger Tip Covers	4	To replace damaged finger tips
6	M5x10mm SHCS	4	For attaching accessories to the base mount points
7	M4x10mm SHCS	4	For attaching accessories to the head mount points
8	M3x14mm Standoffs	4	For attaching accessories to the gripper
9	3ft Ethernet Cable	1	For connecting the robot to the network
10	Fetch Robotics Stickers	5	For your laptop :)
11	Mircofiber Lens Cloth	1	For cleaning optics of the robot

3.1.3 Robot Power Supply

The robot power supply is shown below in the packaging:

The robot plug end is shown below, when removing the charge plug from the robot **always** grab the plug by the housing and not the cable.


fetch
robotics
fetch
robotics
fetch
robotics


1
2
3
4
5
6
7
8
9
10
11
Metric
Eklind
MADE IN USA
fetch
robotics
9000
fetch
robotics
fetch
robotics


HLG-600H-48B
INPUT: 100-240VAC 7.0A 50/60Hz
277VAC 2.9A 50/60Hz
(277VAC for UL8750 only)


fetch

Warning: Pulling on the charge plug cable instead of the handle can cause damage to the cable assembly over time and could potentially cause injury to the robot or user.

3.2 Running Fetch and Freight

3.2.1 Turning on Fetch and Freight

To turn on the robot, set the *Power Disconnect Switch* (the red one on the lower back of the robot) to the ON position and then press the power switch on the access panel until it lights up.

3.2.2 Logging In

Once the robot is turned on and the robot is on the network, ssh into the computer of the robot using the default `fetch` user account:

```
>$ ssh fetch@<robot_name_or_ip>
```

robot_name_or_ip will be either an IPv4 network address, or a network name, depending on the configuration of your local network. If your computer and network is setup for multicast DNS (mDNS) then you may be able to use `fetchXYZ.local` as the network name where XYZ will be the serial number of your robot (remove any leading zeros from the serial number).

3.2.3 Connecting a Monitor (Optional)

If you are unable to access the robot through ssh due to your network settings you will need to connect an HDMI monitor, USB keyboard and USB mouse to the side panel ports. Then use them to select your network from the networking menu, it looks like an empty quarter circle pointed downward when disconnected, in the upper right hand corner of the screen. Once it is set it should remain connected through all reboots.

Warning: Monitor cables that conform to HDMI specifications will fit the robot's monitor port. But some cheaper cables have overmolding that is too large given the spec and may not seat properly. Please ensure the HDMI cable you are using is the correct size.

Note: If you are still having difficulty connecting to your robot you will need to contact your network administrator for specific instructions on how to connect the robot to your network.

Once connected then create your user account as shown below.

Default User Account

Each robot ships with a default user account, with username *fetch* and password *robotics*. It is recommended to change the password when setting up the robot.

Creating User Accounts

It is recommended that each user create their own account on the robot, especially when developing from source. To create an account on the robot, ssh into the robot as the *fetch* user, and run the following commands:

```
>$ sudo adduser USERNAME
>$ sudo usermod -G adm,cdrom,sudo,dip,plugdev,lpadmin,sambashare USERNAME
```

3.2.4 Tucking Fetch's Arm

Note: The tuck arm server is a new feature released in fetch_teleop and fetch_bringup 0.6.0. The deadman became required in fetch_teleop 0.7.0.

To tuck the arm, press and hold button 6, as shown in the image of the controller below, for one second. This will trigger the tuck_arm server to tuck the arm. While tucking the arm, Fetch will avoid collisions with itself, however it will not be using any active perception, so be sure to keep the space in front of the robot clear when running the tuck arm. While the arm is tucking, you will have to hold the deadman. Releasing the deadman will stop the tucking action.

3.2.5 Driving Fetch and Freight with a Joystick

Each Fetch and Freight ship with a robot joystick. Whenever the robot drivers are running, so is joystick teleop. The joystick is capable of controlling the movement of the robot base, torso, head and gripper.

Button #	Function (details below)
0	Open gripper
1	Control robot turning
2	Control forward/backward driving
3	Close gripper
4	Disable motor position holding
5	Not used
6	Arm tuck
7	Not used
8	Head control deadman
9	Not used
10	Primary deadman
11	Not used
12	Torso up
13	Not used
14	Torso down
15	Not used
16	Pair/unpair with robot

To pair the controller with the robot, press the middle button (16) once the robot has powered on. The controller will vibrate once successful. To unpair, hold the button for 10 s. The LED indicator on top will turn off.

To drive the robot base, hold the primary deadman button (button 10 above) and use the two joysticks. The left joystick controls turning velocity while the right joystick controls forward velocity.

Warning: Whenever driving the robot, always lower the torso and tuck the arm to avoid potentially unstable operation.

To control the head, release the primary deadman and hold the head deadman (button 8). The left joystick now controls head pan while the right joystick controls head tilt.

To move the torso up, hold the primary deadman and press the triangle button (12). To move the torso down, hold the primary deadman and press the X (14).

To close the gripper, hold the primary deadman and press the close button (3). To open, hold the primary deadman and

press the open button (0).

Some controllers, such as the arm and head controllers, will attempt to hold position indefinitely. Sometimes this is not desired. Holding button (4) for 1 second will stop all controllers except the base controller and the arm gravity compensation.

Moving the Base with your Keyboard

Note: You will need a computer with ROS installed to properly communicate with the robot. Please consult the ROS Wiki for more information. We strongly suggest an Ubuntu machine with ROS Indigo installed.

To teleoperate the robot base in simulation, we recommend using the `teleop_twist_keyboard.py` script from teleop_twist_keyboard package.

```
>$ export ROS_MASTER_URI=http://<robot_name_or_ip>:11311
>$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

Software Runstop

In addition to the runstop button on the side of the robot, similar software functionality is also available, allowing for button presses on the PS3 controller or a program to disable the breakers. This functionality is available in release 0.7.3 of the fetch_bringup package. The teleop portion is disabled by default.

3.2.6 Using Software Runstop

To activate the software runstop, publish True to the /enable_software_runstop topic.

Alternately, with the teleop runstop enabled, pressing both of the right trigger buttons (buttons 9 and 11) will activate the software runstop. The software_runstop.py script in fetch_bringup can be modified to change the button(s) for the software runstop.

Once activated, the software runstop can be deactivated by (1) toggling the hardware runstop, or (2) disabling the software runstop by passing False to the /enable_software_runstop topic.

3.2.7 Enable Teleop Software Runstop

Note: In order to edit the robot.launch file, you will need to use a terminal editor (such as nano or vim), or use the -X flag with SSH to use a graphical editor (such as gedit). Additionally, the editor must be launched with *sudo*. Instructions below use nano.

To enable the software runstop, first SSH into the robot, and then modify the robot drivers launch file to use it.

We need to modify the robot.launch file to pass the correct arg to the software runstop script:

```
>$ sudo nano /etc/ros/indigo/robot.launch
```

In this file there should be a Software Runstop entry near the end. By default this entry contains an args line, with a value of "-a -b -g". To add teleop control, add the "-t" flag as well. This section will then look like the below. If your robot is an older one and does not have a Software Runstop entry, you will want to simply copy the block the below.

```
<!-- Software Runstop -->
<include file="$(find fetch_bringup)/launch/include/runstop.launch.xml">
  <arg name="flags" value="-a -b -g -t" />
</include>
```

Note that the -a, -b, -g flags correspond to letting the software runstop control the *arm, base and gripper breakers*, respectively.

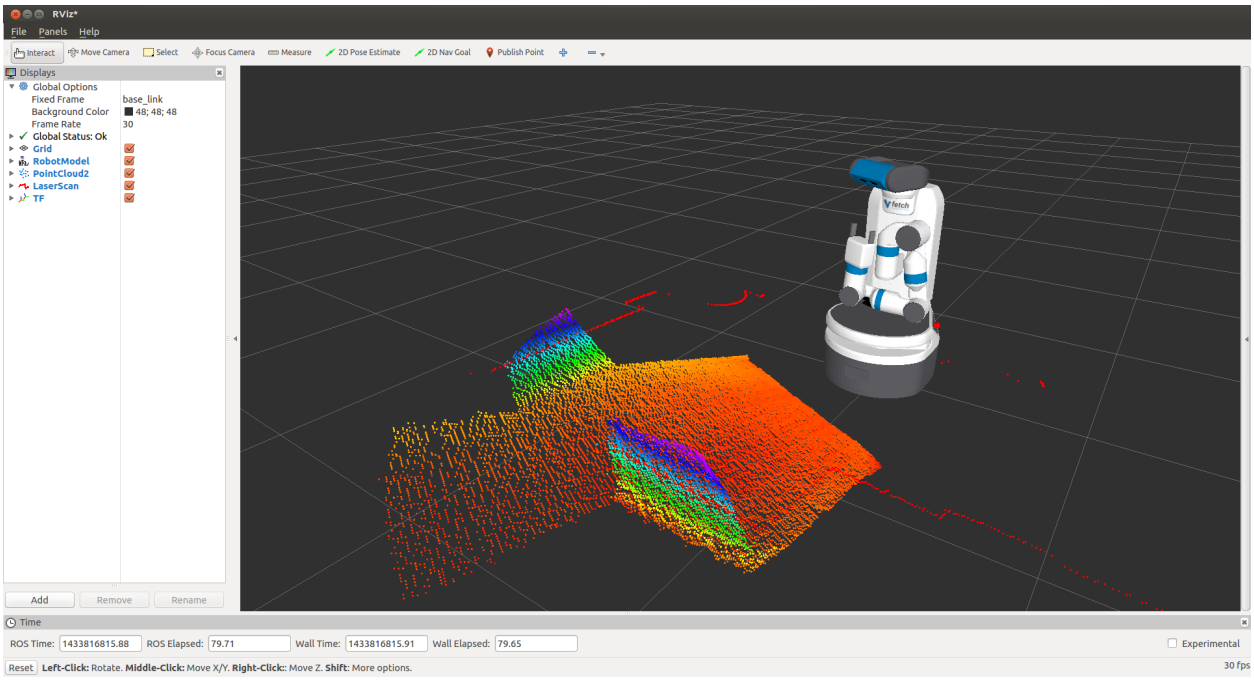
Additionally, if completely disabling the software runstop functionality is desired, the above section in robot.launch can be commented out or removed.

Finally, restart the drivers so that our changes take effect:

```
>$ sudo service robot stop && sudo service robot start
```

3.2.8 Visualizing Data

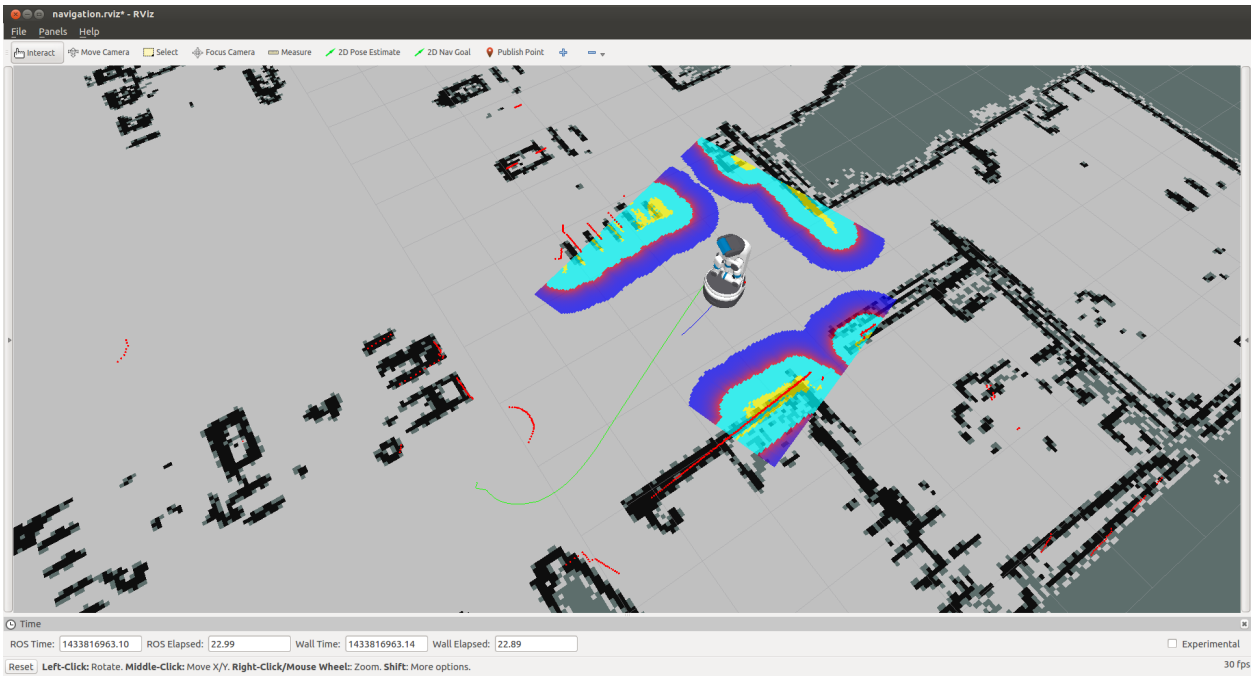
```
>$ export ROS_MASTER_URI=http://<robot_name_or_ip>:11311
>$ rosrun rviz rviz
```



Note: You will need a computer with ROS installed to properly communicate with the robot. Please consult the ROS Wiki for more information. We strongly suggest an Ubuntu machine with ROS Indigo installed.

You can now manually set up your RVIZ visualization or re-run RVIZ with a configuration file using the command line. The default `.rviz` configuration file for Fetch can be loaded using:

```
>$ roscd fetch_navigation/config
>$ export ROS_MASTER_URI=http://<robot_name_or_ip>:11311
>$ rviz -d navigation.rviz
```



Using the Runtime Monitor

Fetch and Freight publish ROS diagnostics messages. These are human-readable messages that inform users of the robot system state. The *runtime_monitor*, part of `rqt_robot_plugins` can be used to view diagnostics from your desktop computer:

```
>$ export ROS_MASTER_URI=http://<robot_name_or_ip>:11311
>$ rosrun rqt_runtime_monitor rqt_runtime_monitor
```

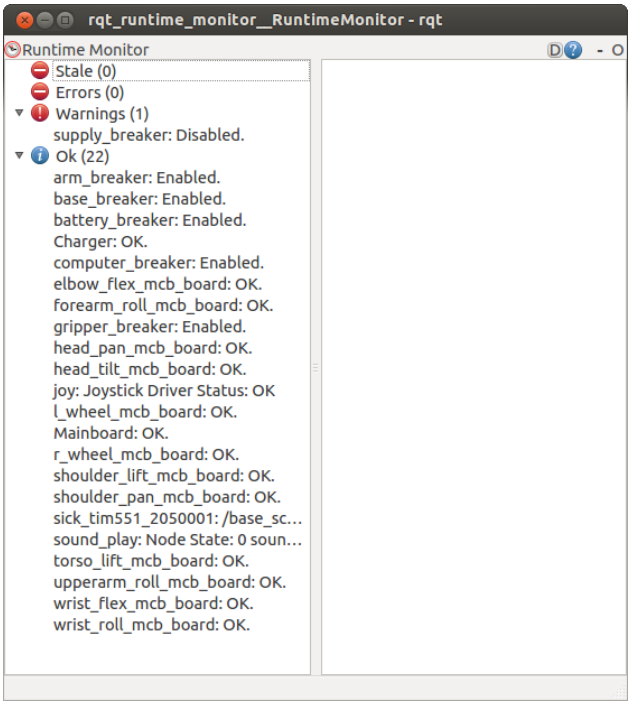
The runtime monitor will have one entry per motor controller board (MCB), as well as one entry per breaker. Each of these entries will be classified as either stale, an error, a warning, or OK. In the above image, the supply_breaker is disabled because the robot is not plugged in – this is only a warning, and not actually an issue.

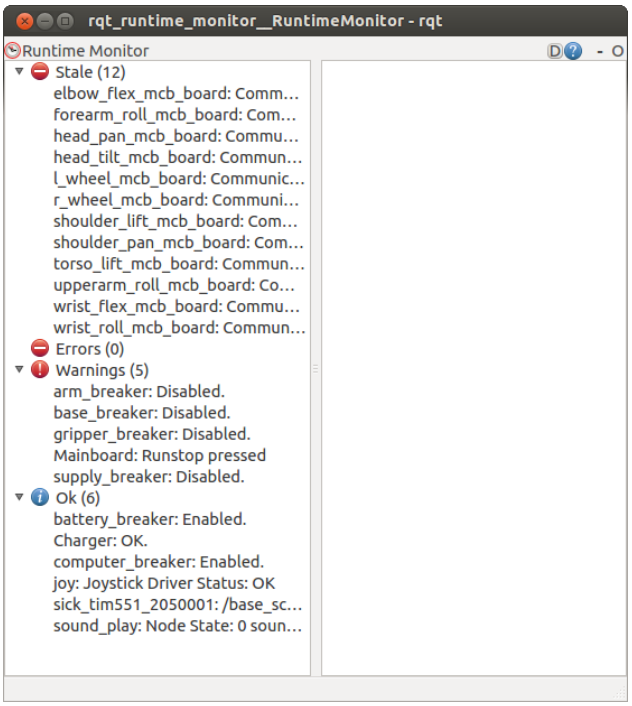
Common errors that can be detected are overly hot motors or breakers, breakers that have tripped. When the runstop on Fetch is pressed, a number of breakers become disabled and the motor controller boards are turned off, causing them to go stale. The below image shows what a runstopped Fetch might look like:

3.2.9 Putting Fetch and Freight Away

Before turning the robot off, it is recommended that you tuck the arm. When power is shut off, the arm will want to slowly fall and will be more difficult to backdrive when in the off configuration.

To turn the robot off, press and hold the illuminated power button on the access panel until it starts blinking. The power button will continue blinking until the computer has successfully shut down, and then power will be disconnected.


rqt_runtime_monitor__RuntimeMonitor - rqt
Runtime Monitor
Stale (0)
Errors (0)
Warnings (1)
supply_breaker: Disabled.
Ok (22)
arm_breaker: Enabled.
base_breaker: Enabled.
battery_breaker: Enabled.
Charger: OK.
computer_breaker: Enabled.
elbow_flex_mcb_board: OK.
forearm_roll_mcb_board: OK.
gripper_breaker: Enabled.
head_pan_mcb_board: OK.
head_tilt_mcb_board: OK.
joy: Joystick Driver Status: OK
l_wheel_mcb_board: OK.
Mainboard: OK.
r_wheel_mcb_board: OK.
shoulder_lift_mcb_board: OK.
shoulder_pan_mcb_board: OK.
sick_tim551_2050001: /base_sc…
sound_play: Node State: 0 soun…
torso_lift_mcb_board: OK.
upperarm_roll_mcb_board: OK.
wrist_flex_mcb_board: OK.
wrist_roll_mcb_board: OK.


rqt_runtime_monitor__RuntimeMonitor - rqt
Runtime Monitor
Stale (12)
elbow_flex_mcb_board: Comm…
forearm_roll_mcb_board: Com…
head_pan_mcb_board: Commu…
head_tilt_mcb_board: Commun…
l_wheel_mcb_board: Communic…
r_wheel_mcb_board: Communi…
shoulder_lift_mcb_board: Com…
shoulder_pan_mcb_board: Com…
torso_lift_mcb_board: Commun…
upperarm_roll_mcb_board: Co…
wrist_flex_mcb_board: Commu…
wrist_roll_mcb_board: Commun…
Errors (0)
Warnings (5)
arm_breaker: Disabled.
base_breaker: Disabled.
gripper_breaker: Disabled.
Mainboard: Runstop pressed
supply_breaker: Disabled.
Ok (6)
battery_breaker: Enabled.
Charger: OK.
computer_breaker: Enabled.
joy: Joystick Driver Status: OK
sick_tim551_2050001: /base_sc…
sound_play: Node State: 0 soun…

CHAPTER

FOUR

ROBOT HARDWARE OVERVIEW

4.1 Mechanism Terminology

The Fetch and Freight kinematics are defined by using the concepts of joints, links, and coordinate frames. The robot URDF (unified robot description format) model specifies the attributes (kinematic tree, names, ranges, etc.) of the joints, links, and frames of the robot. A link element in the URDF describes a rigid body with inertia, visual features, and coordinate frames. A joint element in the URDF defines the kinematics, dynamics, safety limits, and type (revolute, continuous fixed, prismatic, floating, or planar). Fixed joints are typically used to describe the relationship between two rigidly joined components in the robot.

4.1.1 Link

The links for the Fetch and Freight are defined in the URDF description that are located in the fetch_description or freight_description package respectively.

4.1.2 Frame

Frames represent the coordinate frames of links, detected objects, sensors, or the location of another robot in the world. Frames are define relative to other frames and the transformations between each frame is tracked using TF.

4.1.3 Joint

A joints define the relationship between links and are defined in the URDF description that can be found in the fetch_description or freight_description package respectively. In the Fetch and Freight the majority of the joints are rotational, the torso is prismatic, and there are several fixed joints describing the location of sensors within the robot. Rotational and translational joints are represented similarly in the URDF, and joint forces are described as *effort* instead of force or torque. Position, and velocity are both used to describe linear and angular motion of a joint.

4.1.4 Fetch Home Pose

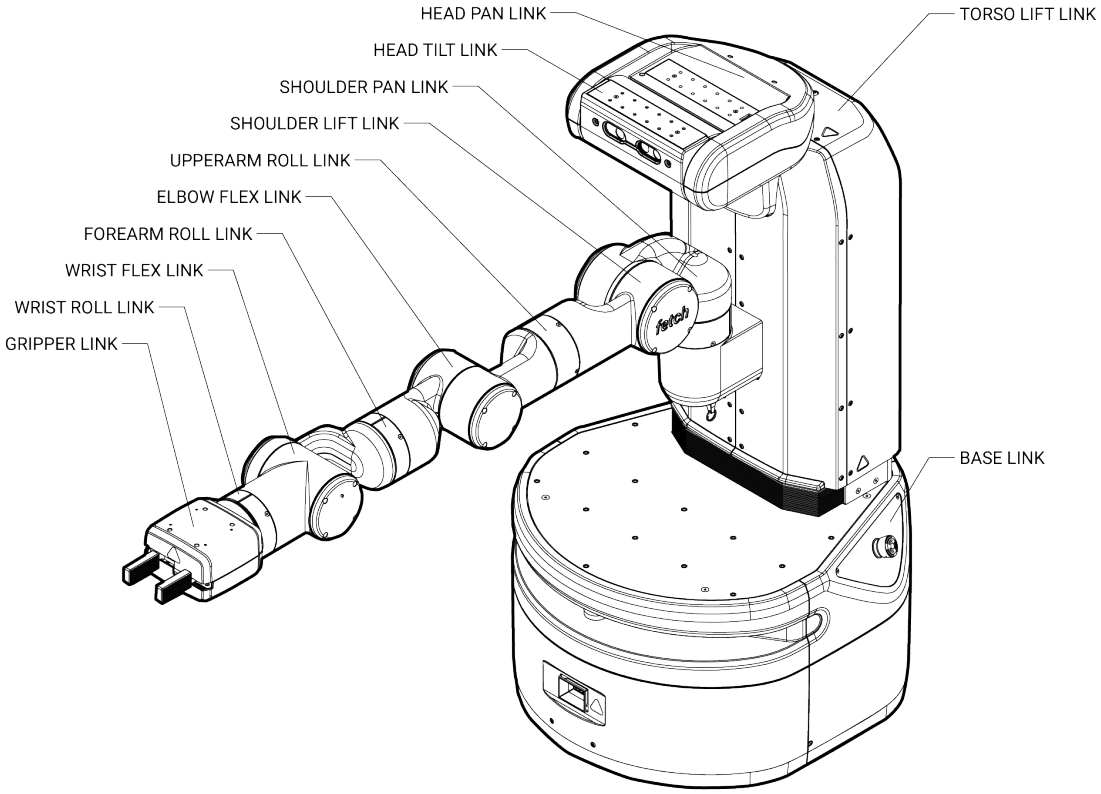
The home pose of the Fetch robot is used to describe the joint positions in a consistent manner. The home pose is defined as the robot arm straight out in front of the robot with all x-axes aligned, the gripper in the closed position, and the x-axes of the head parallel and aligned with the arm, in this position all of the joints are considered to be zeroed. The calibration reference for most joints are not at the joint zero and the URDF of the Fetch contains the offsets for each joint.

4.1.5 Coordinate System

The coordinate frames for all links in the Fetch and Freight are defined with positive z-axis up, positive x-axis forward, and positive y-axis to the robot-left when Fetch is in the home pose. All joint angle conventions are chosen such that from the home pose, positive motion of the joint results in positive motion around a positive axis of a joint coordinate frame (i.e. right handed).

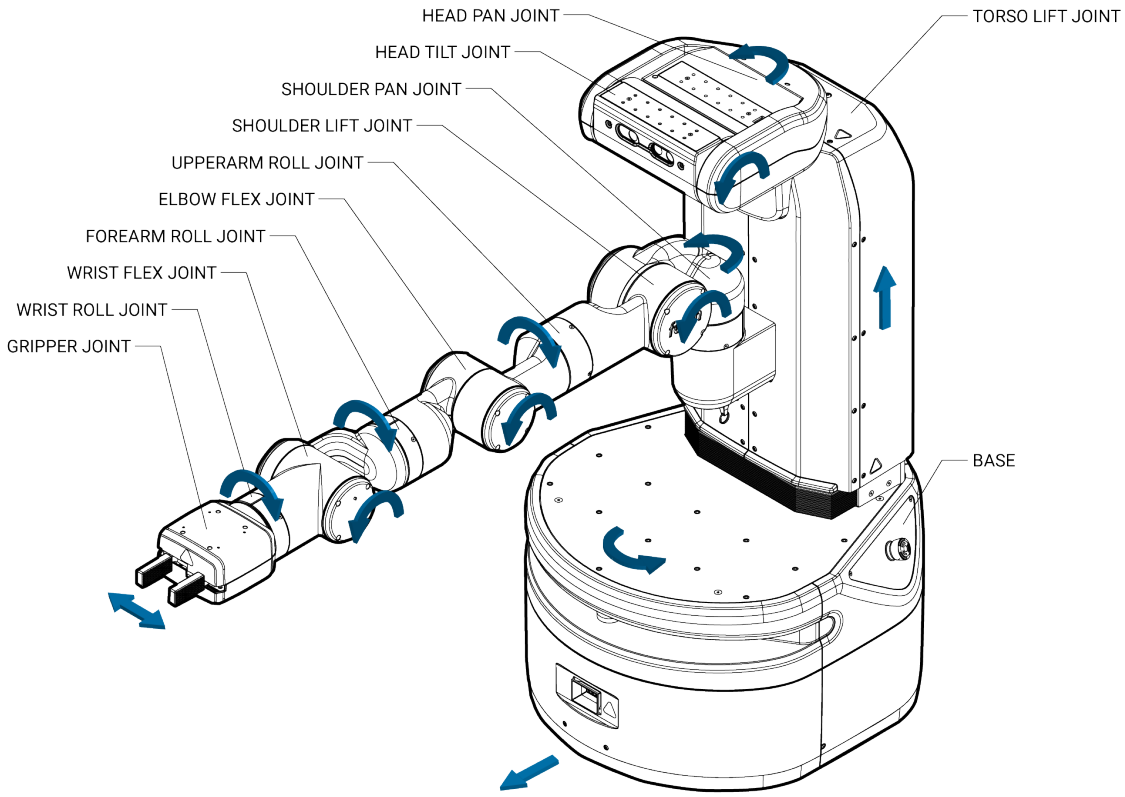
4.1.6 Naming Conventions

In general, the names for a link, a joint, and frame will be similar (e.g. shoulder_pan_link, shoulder_pan_joint, and shoulder_pan_frame). Short prefixes are used to describe the location of repeated components (i.e. drive wheels). The diagrams below show the link and joint naming conventions as well as the positive direction of joint motion.



4.2 Mechanical Overview

Do not operate Fetch or Freight before reviewing the mechanical information listed below.


HEAD PAN JOINT
HEAD TILT JOINT
SHOULDER PAN JOINT
SHOULDER LIFT JOINT
UPPERARM ROLL JOINT
ELBOW FLEX JOINT
FOREARM ROLL JOINT
WRIST FLEX JOINT
WRIST ROLL JOINT
GRIPPER JOINT
TORSO LIFT JOINT
BASE

4.2.1 Environmental

The Fetch and Freight Research Edition Robots are indoor laboratory robots. Operating outside this type of environment could cause damage to the Fetch and Freight robots, and injury or death to operators.

Drive Surface

- The drive surface of the Fetch must be capable of supporting the entire weight of Fetch, about 250 lbs (113.3 kgs). If the surface is too soft, Fetch can get stuck and fail to drive. A commercial carpet or tile is recommended.
- The drive surface of the Freight must be capable of supporting the entire weight of the Freight about 150 lbs (68 kgs) plus the weight of the payload. If the surface is too soft, Freight can get stuck and fail to drive. A commercial carpet or tile is recommended.

Incline Surface

- Fetch and Freight are ready for ADA-compliant ramps, which are at no more than a 1/12 slope. Ramps that are steeper than a 1/12 slope are unsafe and may be a tipping hazard.

Water

- Fetch and Freight have not been tested for any type of contact with water or any other liquid. Under no circumstances should Fetch nor Freight come in contact with water from rain, mist, ground water (puddles) and any other liquid. Water contact can cause damage to the electrical circuitry and the mechanism.

Temperature and Humidity

- Fetch and Freight are designed to run in environments between 15C and 35C.
- Keep Fetch and Freight away from open flames and other heat sources. Never use Fetch or Freight around stoves or ovens.

4.2.2 Forces and Torques

Joint position, velocity, and force limits are implemented in the Fetch and Freight URDF file as well as in firmware. Firmware implements additional power limits. These joint limits control the range of travel of the mechanism and the allowable velocity to prevent over-travel. These limits are enforced by the controller, and are designed to prevent poorly commanded control efforts from damaging the robot or harming operators.

The limits below are from the Fetch and Freight URDF files. If a velocity or torque limit is not specified, no value is enforced.

Joint	Velocity	Torque/Force	Power
"*"_wheel_joint	17.4rad/s	8.85Nm	120W
torso_lift_joint	0.1m/s	450N	TBD
head_pan_joint	1.57rad/s	0.32Nm	TBD
head_tilt_joint	1.57rad/s	0.68Nm	TBD
shoulder_pan_joint	1.25rad/s	33.82Nm	TBD
shoulder_lift_joint	1.45rad/s	131.76Nm	TBD
upperarm_roll_joint	1.57rad/s	76.94Nm	TBD
elbow_flex_joint	1.52rad/s	66.18Nm	TBD
forearm_roll_joint	1.57rad/s	29.35Nm	TBD
wrist_flex_joint	2.26rad/s	25.70Nm	TBD
wrist_roll_joint	2.26rad/s	7.36Nm	TBD
"*"_gripper_finger_joint	0.05m/s	60N	–

4.2.3 Joint Limits and Types

The position limits for Fetch and Freight are specified below. These "hard limits" are the maximum travel for the mechanism.

Joint	Type	Limit (+)	Limit (-)
"*"_wheel_joint	continuous	–	–
torso_lift_joint	prismatic	400mm	0mm
head_pan_joint	revolute	90°	90°
head_tilt_joint	revolute	90° (down)	45° (up)
shoulder_pan_joint	revolute	92°	92°
shoulder_lift_joint	revolute	87°	70°
upperarm_roll_joint	continuous	–	–
elbow_flex_joint	revolute	129°	129°
forearm_roll_joint	continuous	–	–
wrist_flex_joint	revolute	125°	125°
wrist_roll_joint	continuous	–	–
"*"_gripper_finger_joint	prismatic	50mm	0mm

4.2.4 Mount Points

Fetch and Freight both have mount points on top of the robot base. Fetch also has mount points on the head and gripper. The table below lists the fasteners that should be used and the max torques to use when installing fasteners.

Location	Fastener	Max Torque
Base	M5*	8.5Nm
Head	M4	4.2Nm
Gripper	M3	0.7Nm (into the standoff)

Note: When mounting additional items on top of the base, a 4mm maximum thread engagement into the mounting holes must be maintained. Exceeding this maximum may cause the top plate of the base to deflect.

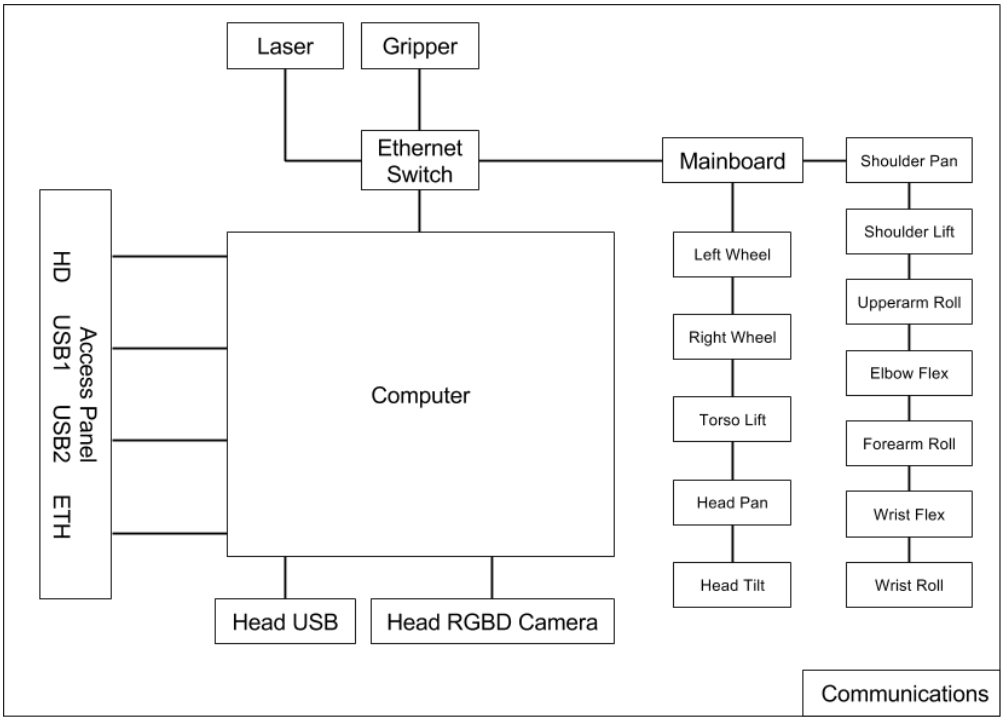
4.2.5 Gripper Modularity Interface

The gripper interface is modular, allowing the gripper to be replaced with alternate configurations. The gripper interface is based on an ISO mechanical standard, Ethernet communications and 24V power. For further details, contact Fetch Robotics for the *Gripper Interface Specification* document.

4.3 Electrical Overview

Internally, Fetch and Freight have a number of circuit boards, communication buses, and other components which handle power distribution and motion control. The system comprises:

- The robot computer, running ROS, sends commands to the mainboard and gripper board over an Ethernet interface. This same Ethernet interface is used to communicate with the scanning laser range finder in the base of the robot. The `fetch_drivers` package provides an interface from the robot computer to the mainboard, and the default gripper.
- The mainboard then communicates with the various motor controller boards (MCB) located throughout the robot. Communication with the MCBs is done over several half-duplex RS-485 buses. In addition to communications-related tasks, the mainboard also controls a number of electronic circuit breakers and also carries the charger circuitry which charges the batteries.
- Each joint has a dedicated MCB with a dedicated microcontroller and an RS-485 connection. The arm MCBs all share one bus while all other MCBs share the other RS-485 bus. The microcontroller on each MCB is where the real-time control loops run.



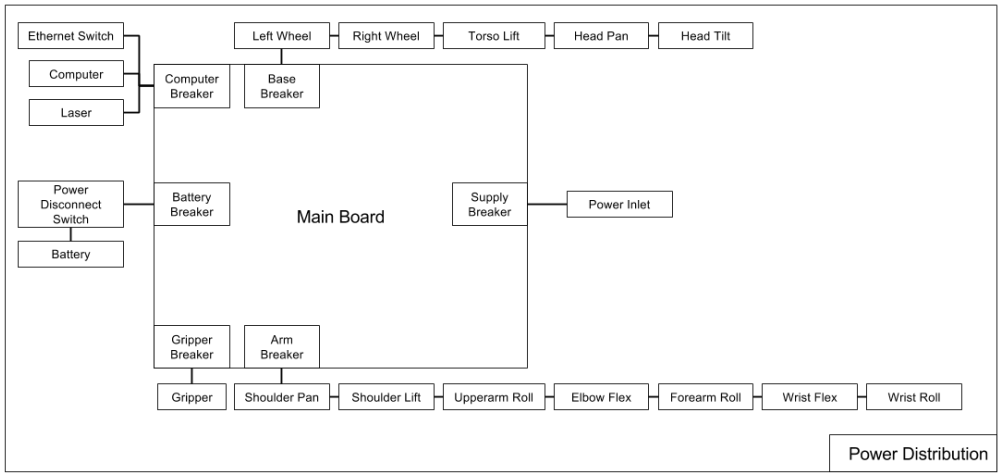
Fetch and Freight both have two 12V Sealed Lead Acid (SLA) batteries located in the robot base. The batteries are connected in series, providing the nominal 24V power rail for the robot. These batteries are kept charged by the mainboard (see *Charging*).

4.3.1 Breakers

There are several breakers within the robot. These are designed in order to prevent damage to the robot if cabling should become worn or shorted out, or in the case of sudden, unexpected overload of a joint. The table below describes each breaker, using the names that are used in `fetch_drivers` and diagnostics:

Breaker Name	Usage
supply_breaker	Limits current between the charging inlet and mainboard.
battery_breaker	Limits current between the mainboard and batteries.
computer_breaker	Limits current delivered to the robot computer.
base_breaker	Limits current delivered to base MCBs, as well as torso and head MCBs on Fetch.
arm_breaker	Limits current delivered to the MCBs located in the arm
gripper_breaker	Limits current delivered to the gripper.

When a breaker is disabled or tripped, power will no longer flow to the connected devices. In the case of MCBs, this means that they will not be able to communicate with the mainboard.



4.3.2 Power Disconnect Switch

The power disconnect is on the lower back of the robot. This switch cuts the power between the battery and the mainboard. It also acts as a breaker, limiting the total current that can be delivered by the batteries.

4.3.3 Runstop

The runstop is used to stop all operation of the joints. It works by disabling the base, arm and gripper breakers. When the runstop is pressed, the drivers will not be able to communicate with the MCBs, and thus their position and other data will not update in RVIZ nor the runtime monitor.

4.3.4 Access Panel

Fetch and Freight both have an access panel with 2 USB, an Ethernet, and an HD Video port. All of these ports are connected directly to the onboard computer. In addition, Fetch has an extra USB port on the head.

Item #	Item Name
1	HD Video Port
2	USB Port 1
3	USB Port 2
4	Ethernet Port
5	Power Button
6	Charge Indicator Light


O
I

The access panel is also the location of the power button which turns the robot on or off. This switch is connected to the mainboard and will only work if the power disconnect switch (the red one on the lower back of the robot) is in the ON position. Pressing the power button until it lights up with boot the robot, including the computer. To turn the robot off, press and hold the illuminated power button on the access panel until it starts blinking. The button will continue blinking until the computer has successfully shut down, and then power will be disconnected.

4.4 Motion Control

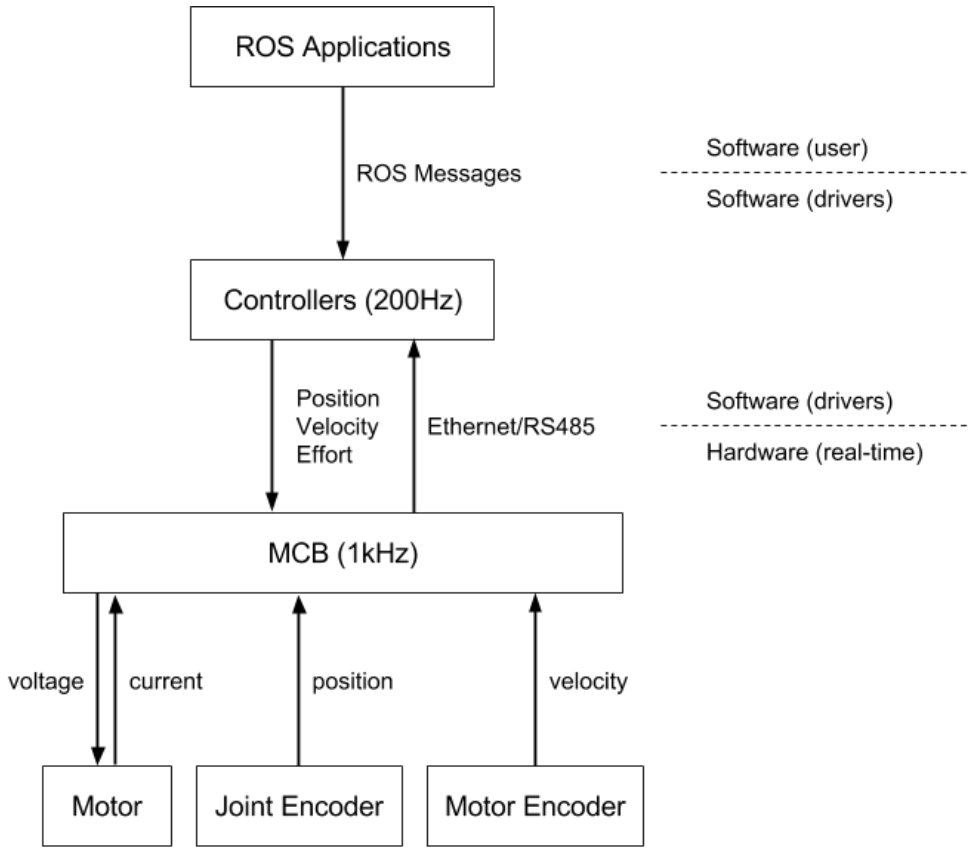
Each joint has a dedicated motor controller board (MCB) with a dedicated microcontroller. The real-time components of the controls run on the MCBs, while the robot computer streams commands to the MCBs.

All motors are brushless and each MCB runs an effort controller at 17.5kHz. Each MCB can also run an optional velocity controller which takes a desired velocity and outputs a control into the effort controller. Further, each MCB can run a position controller which can feed into the velocity controller. The position and velocity controllers each run at 1kHz, and the default rate for the commands streaming from the robot computer is 200Hz. Each MCB can then receive any of the following types of commands:

- Desired position, desired velocity, and desired effort
- Desired velocity and desired effort
- Desired effort

Each MCB returns the measured effort, velocity and position of the joint it is controlling.

Most users will want to use the high-level *Arm and Torso* ROS API, however, users can also create their own controllers as ROS plugins that run on the robot computer and send commands to the MCBs using the robot_controllers_interface



package.

4.5 Sensor Overview

4.5.1 Base Laser

Both Fetch and Freight have a SICK TIM571 scanning range finder. The laser has a range of 25m, 220° field of view, 15Hz update rate and angular resolution of 1/3°. The laser publishes both distance and RSSI (Received Signal Strength Indication) to the **base_scan** topic.

4.5.2 IMU

The mainboard of Fetch and Freight have a 6-axis inertial measurement unit (IMU). The gyroscope within the IMU is capable of measuring +/-2000 degrees per second, while the accelerometers are capable of measuring +/-2g. See *IMU Interface* for details on the ROS API.

4.5.3 Head Camera

Fetch has a Primesense Carmine 1.09 short-range RGBD sensor. This camera is best calibrated in the 0.35-1.4m range. See *Head Camera Interface* for details on the ROS API.

4.5.4 Gripper Sensors

In addition to the position and effort feedback of the gripper joint, the gripper incorporates a 6-axis inertial measurement unit (IMU). The gyroscope within the IMU is capable of measuring +/-2000 degrees per second, while the accelerometers are capable of measuring +/-2g. See *IMU Interface* for details on the ROS API.

CHAPTER

FIVE

COMPUTER OVERVIEW AND CONFIGURATION

Both Fetch and Freight have an internal computer which runs a Long Term Support (LTS) release of Ubuntu and an LTS release of ROS. These releases are intended to give long-term stability to the system.

5.1 Default User Account

Each robot ships with a default user account, with username *fetch* and password *robotics*. It is recommended to change the password when setting up the robot.

5.2 Creating User Accounts

It is recommended that each user create their own account on the robot, especially when developing from source. To create an account on the robot, ssh into the robot as the *fetch* user, and run the following commands:

```
>$ sudo adduser USERNAME
>$ sudo usermod -G adm,cdrom,sudo,dip,plugdev,lpadmin,sambashare USERNAME
```

5.3 Networking

The robot has both internal and external Ethernet-based networks, as well as an external wireless network interface. The external network interfaces are intended for users to connect to the robot, while the internal networks are used to send data between the internal components of the robot.

The majority of communication between components onboard Fetch and Freight happen via the internal Ethernet network. This network is located in the 10.42.42.0/24 subnet and connects the robot computer to the devices listed in the table below. As such, it is important that your building networks do not use the same subnet.

Device	IP Address
Laser range finder	10.42.42.10
Mainboard	10.42.42.42
Gripper	10.42.42.43

There are two possible interfaces for connecting to the robot computer: the wireless interface and the wired interface. Most users will prefer to use the wireless interface, however the access panel also includes a Gigabit Ethernet interface for stationary tasks that require higher bandwidth.

Warning: Never drive the robot with an Ethernet cable attached to the access panel.

5.4 Connecting the Robot to a Wireless Network

The easiest way to configure the wireless networking is to connect a monitor, keyboard, and mouse and use the Network Manager interface.

5.5 Clock Synchronization

It is recommended to install the chrony NTP client on both robots and desktops in order to keep their time synchronized. By default, robots do not ship with chrony installed. To install chrony on Ubuntu:

```
> sudo apt-get update
> sudo apt-get install chrony
```

5.6 Upstart Services

Fetch and Freight use upstart to start and manage various services on the robot. The following upstart services start when the robot is booted:

Name	Description
roscore	starts a roscore
robot	starts robot drivers, requires roscore
sixad	driver for robot joystick over bluetooth
soundplay	starts the sound_play node for audio in ROS

Upstart service can be restart with the *service* command. For instance, to restart the robot drivers:

```
>sudo service robot stop
>sudo service robot start
```

Since the roscore runs independently of the drivers, the drivers can be restarted without having to restart remote instances of RVIZ or similar ROS tools. Note that this also means the parameter server will not be reset when restarting the drivers, and so a roscore restart may be required if the parameter server has been corrupted by a user script.

5.7 Log Files

A number of log files are created on the robot. Log files related to upstart services can be found in the /var/log/upstart folder, the name of the log will be service.log.

ROS logs for the robot and roscore upstart services will be created in the /var/log/ros folder.

5.8 Speakers and Audio

The mainboard of Fetch and Freight contains a USB audio device. While the device enumerates as a standard Linux audio device, we recommend using the sound_play ROS package to access the speakers. `sound_play` is automatically started as an *upstart service* when the robot starts. This service is pre-configured to have the correct group-level access to the audio system. If using the speakers directly through a Linux interface, be sure to add your user to the `audio` group in order to actually access the speakers.

While the `sound_play` ROS interface allows users to set an audio level, the audio level set is a percentage of the audio level set for Linux. To adjust the Linux audio level, use the following command and follow the on-screen instructions:

```
>$ sudo su ros -c "alsamixer -c 1"
```

CHAPTER

SIX

CARE AND FEEDING

6.1 Charging

The robot can be charged by plugging the supply connector of the power brick into the front of the robot or by docking with the robot charge dock. When using the supply connector, make sure the “Fetch” logo is facing up, otherwise the connector will not properly mate:

The connector needs roughly 100 newtons (24 lbs) of force to plug in. If it seems the connector will not mate with this much force, check the plug and connector for damage or foreign objects.

The charge indicator light on the *Access Panel* indicates the status of the battery and charging:

Charge Indicator	Meaning
Solid Green	Robot is fully charged
Blinking green	Robot is charging
Solid Red	Battery voltage is low
Blinking red	Charging error has occurred

The charge indicator light will flash green when the robot is charging. When the robot is completely charged the indicator light will stay solid green.

The charge indicator light will turn red when the battery voltage is low. When the charge indicator turns red, you should recharge the robot. If the battery voltage gets low enough, the robot will automatically turn off.

In some cases, the charging circuit may detect an error condition and will flash the charging indicator red. After about 1 minute, the charger will attempt charging again. If the error condition persists, a support ticket should be created to address the problem.

Note: One notable charging error is caused by the *Power Disconnect Switch* being switched off. When the power disconnect is switch off, it disconnects the battery which prevents the charger from working. When the charger detects this situation it will flash both the charge indicator red, and the power button at the same time.

The robot will typically need about 3.5 hours to charge batteries from empty to 90% capacity. It will take an additional 3 hours to charge remaining 10% of battery capacity. While the robot does not need to get to 100% on every charge, it is strongly recommended that the robot be fully charged at least once a week to get the best possible life out of the batteries. Charging works whether the robot is on or off. It is perfectly acceptable to have robot move its arm, torso, and head while charging. However, you should not drive the robot while charging because it is easy to misjudge the cable length and damage the wiring or connector.

There are also some important rules that should be followed to prevent damage to robot and charging supply.

1. When disconnecting the supply from robot, grab plug and pull out. Never pull on the cable to pull out the plug. Also, when moving power supply brick, do not use cable to pull it around. All copper wire has poor tensile strength and can break with too much abuse.
2. Pull straight back when disconnecting the supply plug, do not pull out at an angle. The connector is not designed to support large sideways forces. It is easy to pull in the correct direction when grabbing the plug, but not when pulling on the cable. This is another reason that the plug should never be pulled out using the cable.
3. Do not attempt to drive robot while plugged into charger. In our experience it is too easy to misjudge the cable length. The robot is heavy and the drive motors can produce high torques. The robot can easily rip out the connector and cord even while moving at slow speeds. Even when turning in place, its possible to pull on the plug if the robot happens to drive over the cord.

If the supply wiring or connector is damaged, do not attempt use the damaged supply. Instead create a support ticket to get the damage part fixed or replaced.

The robot uses large sealed lead-acid (SLA) batteries. These batteries should provide a many hours of robot run time. Like most chemical batteries technologies, the amount energy the batteries can provide will decrease as they age. Luckily, replacing the batteries is simple and relatively inexpensive. Even so, there are few things that can done to improve battery lifetime.

1. Keep batteries charged. Leaving SLA batteries in partially discharged state will age the batteries more quickly, reduce useful running time.
2. Avoid deep discharge of batteries when not necessary. Battery will age less when discharged twice to 50% capacity than discharged once to 100% capacity once.
3. Fully charge battery instead of just partially charging battery between uses. While this is not always feasible, it will slow the rate at which individual battery cells get out of balance.

4. When doing heavily cycling of batteries, leave the batteries charging for an extra long period of time, at least once a week. Even though the batteries may be fully charged, leaving the batteries on charger for longer period of time will help balance out differences in individual cell voltages that accumulate over time. This not as important when batteries are only discharged small amounts during each use.
5. Keep robot cool. Cool batteries age less than warm ones.

6.2 Updating Your Robot

Your robot has been pre-configured with ROS Indigo and the appropriate APT repositories from which to fetch package updates. Upgrading to the latest packages is as easy as:

```
sudo apt-get update
sudo apt-get install --only-upgrade ros-indigo-* f.*-system-config
sudo service robot stop
sudo service robot start
```

Warning: Using 'apt-get upgrade' and 'apt-get dist-upgrade' could cause critical software, such as the kernel, to change. We can not guarantee your robot will function after making such a change. We recommend against using these commands unless you understand and accept the risks.

Each circuit board within the robot is equipped with a bootloader, allowing new and updated firmware to be installed. New releases of the *fetch-drivers* package may include updated firmware for your robot, which will automatically be installed when the drivers are next started (typically by the robot upstart service). When restarting the robot service, there may be a slight delay before the drivers are fully operational if a new firmware upgrade is included.

6.3 Re-Setting up apt-get Sources

If someone has changed or deleted the default apt-get sources then the following commands will create sources.list files so that the robot can see the public ros package server and the Fetch Robotics package server.

```
>$ sudo sh -c 'echo "deb http://packages.fetchrobotics.com/ros/ubuntu trusty main" > /etc/apt/sources

>$ sudo sh -c 'echo "deb http://packages.fetchrobotics.com/ros/ubuntu trusty main" > /etc/apt/sources
```

6.4 Cleaning Your Robot

To clean fingerprints, dirt, and smudges from the skin of Fetch and Freight use a clean soft cloth and isopropyl alcohol or window cleaner (e.g. Windex). Make sure to wet the cloth with the isopropyl alcohol or window cleaner, and then gently clean the skins of the robot.

Warning: Do not spray or pour isopropyl alcohol or window cleaner directly on the skins of the robot, this may damage the skins or worse cause fluids to enter the robot.

To clean the sensor optics of Fetch and Freight use the lens cloth provided in the tool kit. Lens tissues or cotton swabs are also good options for cleaning the optics of the robot.

Warning: Do not use window cleaner, acetone, or abrasive cloths on the sensor lenses as this may cause damage to the lens.

CHAPTER

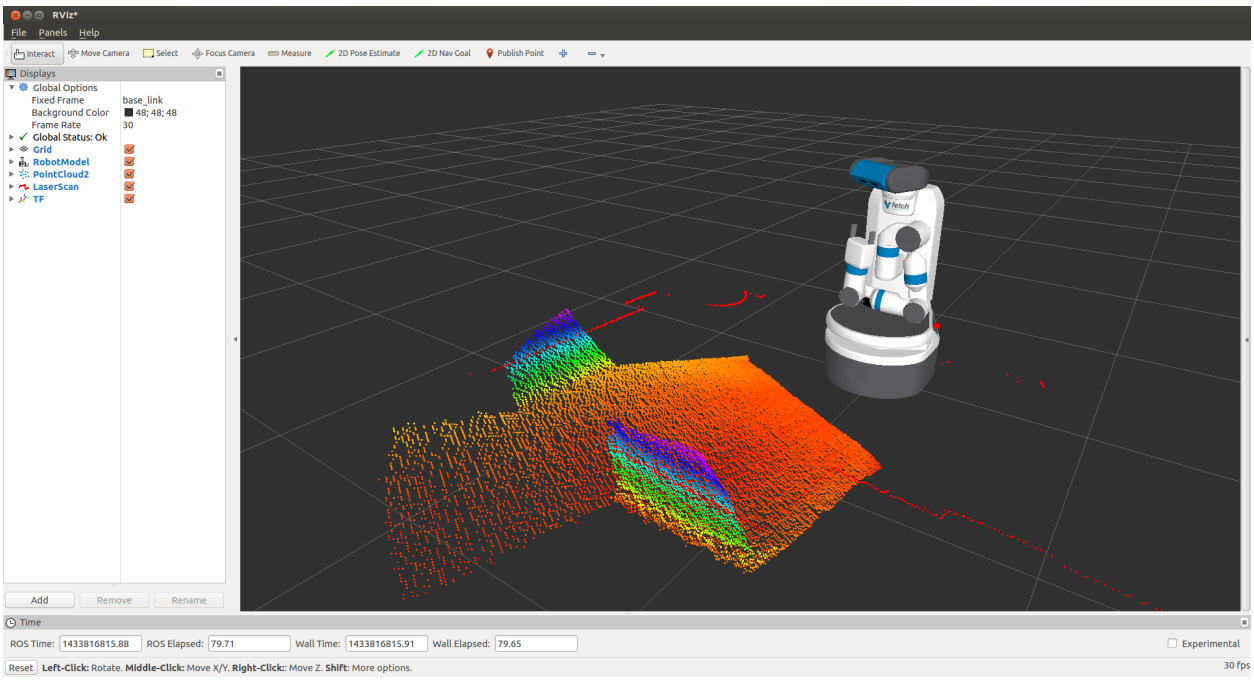
SEVEN

TUTORIALS

7.1 Tutorial: Visualization

7.1.1 Visualizing with RVIZ

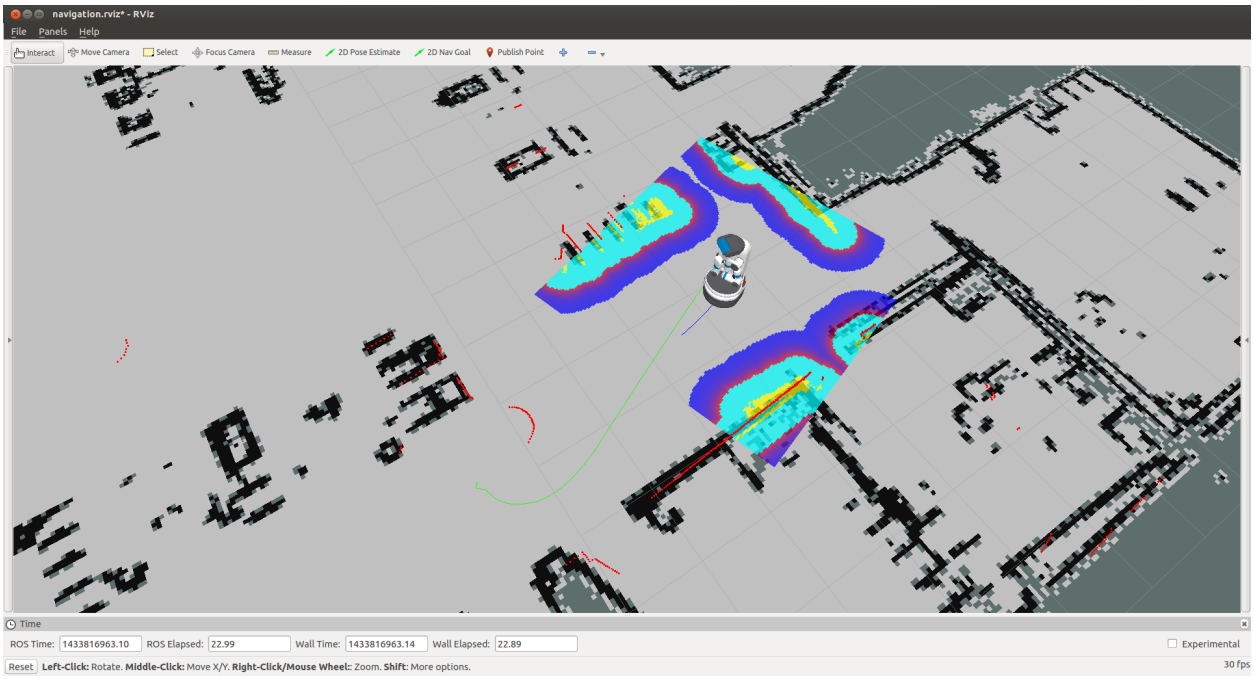
```
>$ export ROS_MASTER_URI=http://<robot_name_or_ip>:11311
>$ rosrun rviz rviz
```



Note: You will need a computer with ROS installed to properly communicate with the robot. Please consult the ROS Wiki for more information. We strongly suggest an Ubuntu machine with ROS Indigo installed.

You can now manually set up your RVIZ visualization or re-run RVIZ with a configuration file using the command line. The default `.rviz` configuration file for Fetch can be loaded using:

```
>$ roscd fetch_navigation/config
>$ export ROS_MASTER_URI=http://<robot_name_or_ip>:11311
>$ rviz -d navigation.rviz
```



7.1.2 Using the Runtime Monitor

Fetch and Freight publish ROS diagnostics messages. These are human-readable messages that inform users of the robot system state. The *runtime_monitor*, part of `rqt_robot_plugins` can be used to view diagnostics from your desktop computer:

```
>$ export ROS_MASTER_URI=http://<robot_name_or_ip>:11311
>$ rosrun rqt_runtime_monitor rqt_runtime_monitor
```

The runtime monitor will have one entry per motor controller board (MCB), as well as one entry per breaker. Each of these entries will be classified as either stale, an error, a warning, or OK. In the above image, the supply_breaker is disabled because the robot is not plugged in – this is only a warning, and not actually an issue.

Common errors that can be detected are overly hot motors or breakers, breakers that have tripped. When the runstop on Fetch is pressed, a number of breakers become disabled and the motor controller boards are turned off, causing them to go stale. The below image shows what a runstopped Fetch might look like:

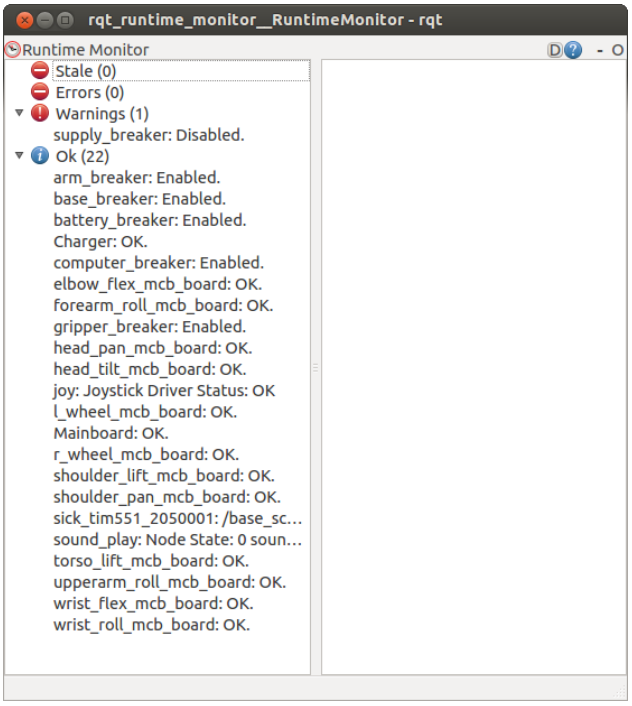
7.2 Tutorial: Gazebo Simulation

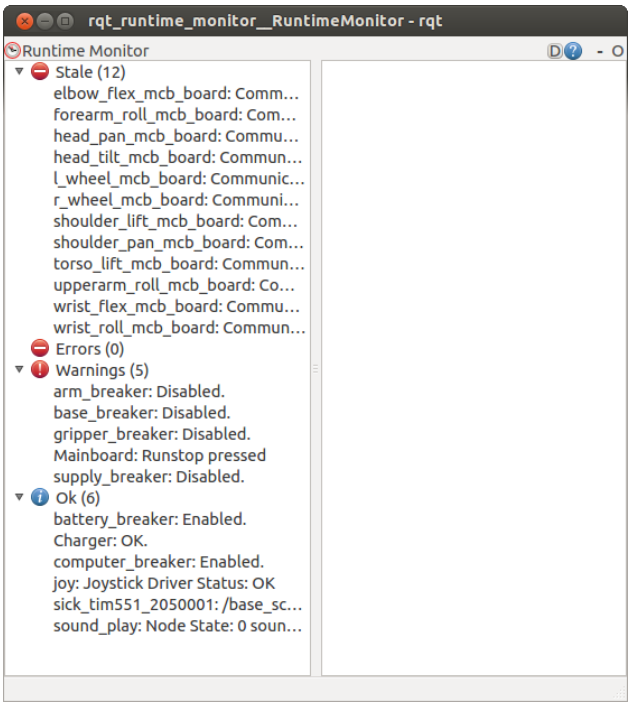
Fetch and Freight have simulated counterparts using the Gazebo Simulator.

7.2.1 Installation

Before installing the simulation environment, make sure your desktop is setup with a standard installation of ROS Indigo on Ubuntu 14.04. Once your APT repositories are configured, you can install the simulator:

```
>$ sudo apt-get update
>$ sudo apt-get install ros-indigo-fetch-gazebo-demo
```


rqt_runtime_monitor__RuntimeMonitor - rqt
Runtime Monitor
Stale (0)
Errors (0)
Warnings (1)
supply_breaker: Disabled.
Ok (22)
arm_breaker: Enabled.
base_breaker: Enabled.
battery_breaker: Enabled.
Charger: OK.
computer_breaker: Enabled.
elbow_flex_mcb_board: OK.
forearm_roll_mcb_board: OK.
gripper_breaker: Enabled.
head_pan_mcb_board: OK.
head_tilt_mcb_board: OK.
joy: Joystick Driver Status: OK
l_wheel_mcb_board: OK.
Mainboard: OK.
r_wheel_mcb_board: OK.
shoulder_lift_mcb_board: OK.
shoulder_pan_mcb_board: OK.
sick_tim551_2050001: /base_sc…
sound_play: Node State: 0 soun…
torso_lift_mcb_board: OK.
upperarm_roll_mcb_board: OK.
wrist_flex_mcb_board: OK.
wrist_roll_mcb_board: OK.


rqt_runtime_monitor__RuntimeMonitor - rqt
Runtime Monitor
Stale (12)
elbow_flex_mcb_board: Comm…
forearm_roll_mcb_board: Com…
head_pan_mcb_board: Commu…
head_tilt_mcb_board: Commun…
l_wheel_mcb_board: Communic…
r_wheel_mcb_board: Communi…
shoulder_lift_mcb_board: Com…
shoulder_pan_mcb_board: Com…
torso_lift_mcb_board: Commun…
upperarm_roll_mcb_board: Co…
wrist_flex_mcb_board: Commu…
wrist_roll_mcb_board: Commun…
Errors (0)
Warnings (5)
arm_breaker: Disabled.
base_breaker: Disabled.
gripper_breaker: Disabled.
Mainboard: Runstop pressed
supply_breaker: Disabled.
Ok (6)
battery_breaker: Enabled.
Charger: OK.
computer_breaker: Enabled.
joy: Joystick Driver Status: OK
sick_tim551_2050001: /base_sc…
sound_play: Node State: 0 soun…

Warning: Never run the simulator on the robot. Simulation requires that the ROS parameter use_sim_time be set to true, which will cause the robot drivers to stop working correctly. In addition, be sure to never start the simulator in a terminal that has the ROS_MASTER_URI set to your robot for the same reasons.

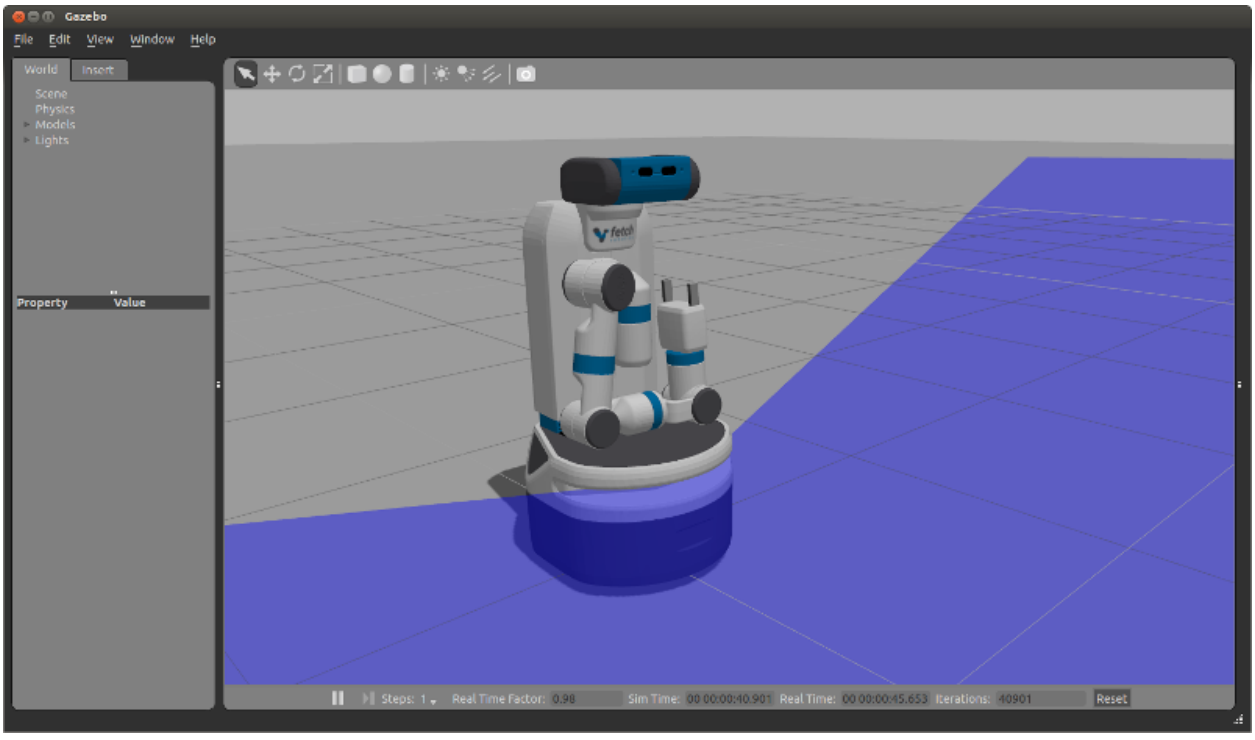
7.2.2 Starting the Simulator

The `fetch_gazebo` and `fetch_gazebo_demo` packages provide the Gazebo environment for Fetch. `fetch_gazebo` includes several launch files:

- simulation.launch spawns a robot in an empty world.
- playground.launch spawns a robot inside a lab-like test environment. This environment has some tables with items that may be picked up and manipulated. It also has a pre-made map which can be used to test out robot navigation and some simple demonstrations of object grasping.

To start the simplest environment:

```
>$ roslaunch fetch_gazebo simulation.launch
```



Note that all of the environments will prepare the robot by tucking the arm and giving the head an initial command.

7.2.3 Simulating a Freight

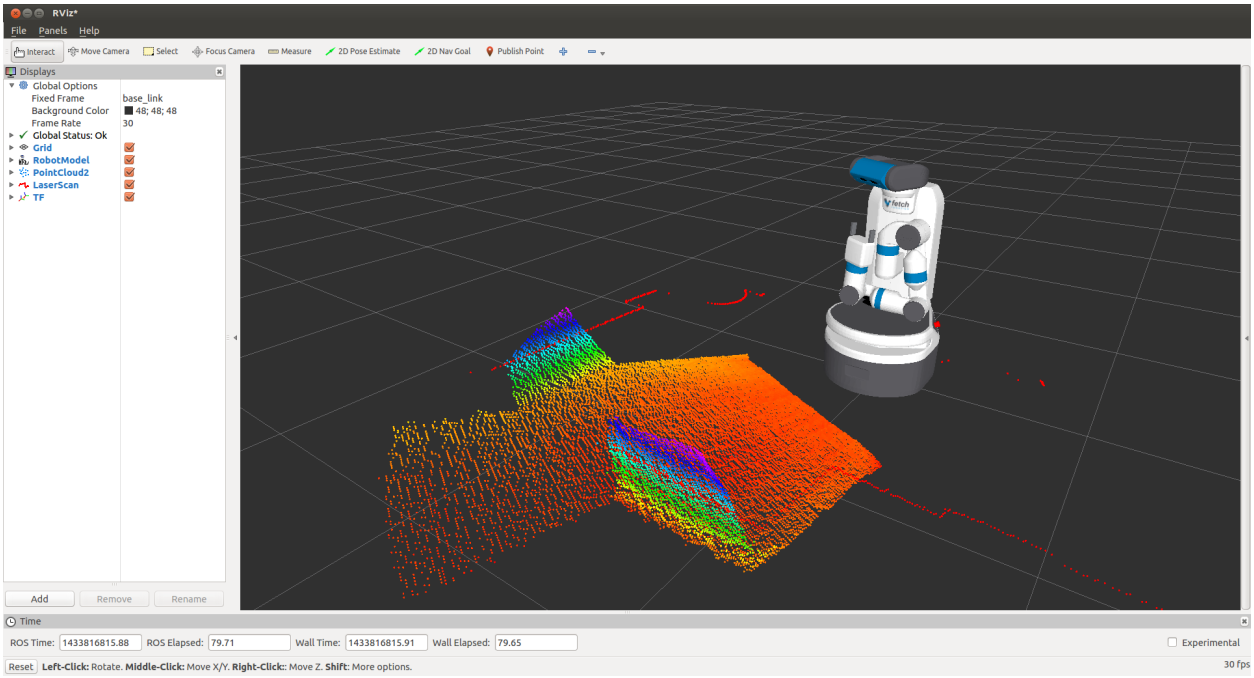
Freight uses the same launch files as Fetch, simply pass the robot argument:

```
>$ roslaunch fetch_gazebo simulation.launch robot:=freight
```

7.2.4 Visualizing with RVIZ

Even though Gazebo has a graphical visualization, RVIZ is still the preferred tool for interacting with your robot.

```
>$ export ROS_MASTER_URI=http://<robot_name_or_ip>:11311
>$ rosrun rviz rviz
```



Note: You will need a computer with ROS installed to properly communicate with the robot. Please consult the ROS Wiki for more information. We strongly suggest an Ubuntu machine with ROS Indigo installed.

You can now manually set up your RVIZ visualization or re-run RVIZ with a configuration file using the command line. The default `.rviz` configuration file for Fetch can be loaded using:

```
>$ roscd fetch_navigation/config
>$ export ROS_MASTER_URI=http://<robot_name_or_ip>:11311
>$ rviz -d navigation.rviz
```

7.2.5 Running the Mobile Manipulation Demo

There is a fully integrated demo showing navigation, perception and MoveIt! working together on the robot in simulation. To run the demo, start Gazebo simulator with the playground:

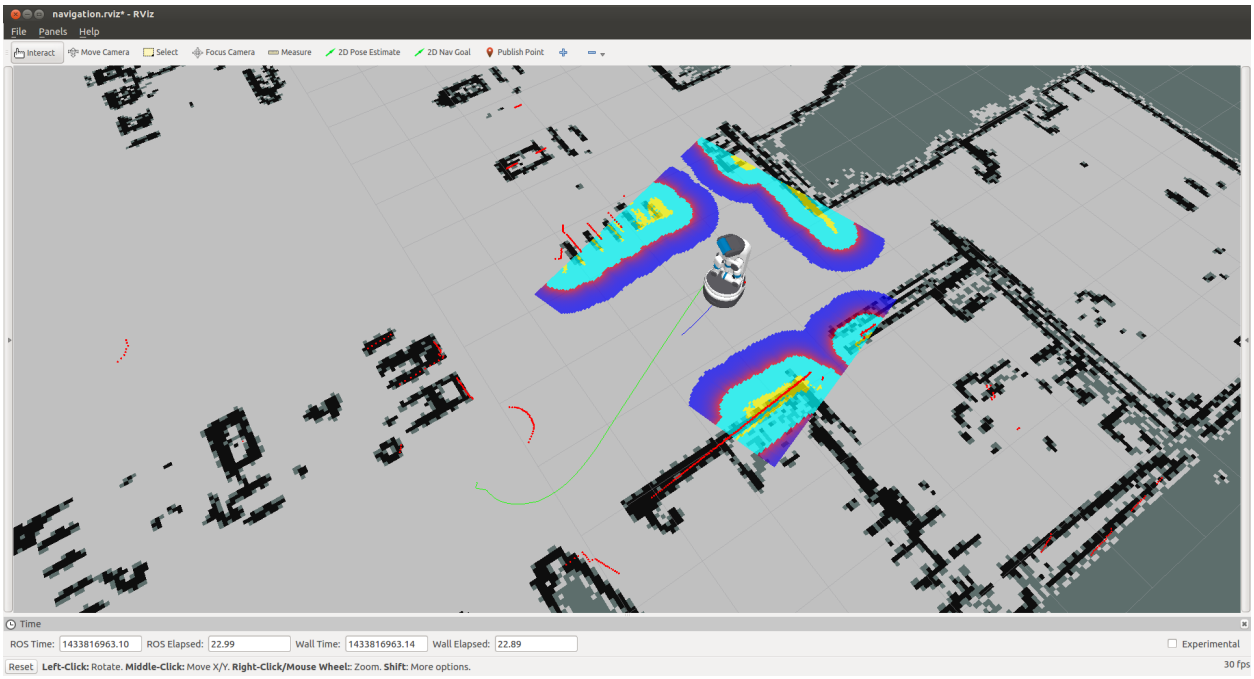
```
>$ roslaunch fetch_gazebo playground.launch
```

Wait until the simulator is fully running and then run the demo launch file:

```
>$ roslaunch fetch_gazebo_demo demo.launch
```

This will start:

- fetch_nav.launch - this is the navigation stack with a pre-built map of the environment.
- move_group.launch - this is the MoveIt configuration which can plan for the movement of the arm.



- basic_grasping_perception - this is a simple demo found in the `simple_grasping` package which segments objects on tables and computes grasps for them.
- demo.py - this our specific demo which navigates the robot from the starting pose in Gazebo to the table, raises the torso, lowers the head to look at the table, and then runs perception to generate a goal for MoveIt. The arm will then grasp the cube on the table, tuck the arm and lower the torso. Once the robot is back in this tucked configuration, the navigation stack will be once again called to navigate into the room with the countertop where the robot will place the cube on the other table.

7.2.6 Simulation vs. Real Robots

The simulated robot may not be identical to the real robot. In fact, the real robot is likely quite a bit better behaved. Also:

- The simulator does not include the IMU. Therefore, there is also no base odometry fusion with IMU data, and the base_controller directly publishes all required TF data directly.
- The simulated robot does not have the head_camera/depth/* topics due to limitations within the Gazebo plugins.
- The simulated robot arm is not as well tuned as the real robot. The real arm will not wobble the way the simulated arm does when executing a trajectory. The simulated robot has also not been tuned with various payloads. It is best used for examining the workspace of the robot, and not the actual controls-related performance of the arm.
- The fingers of the real robot gripper are driven by the same leadscrew, so the object will always be grasped in the center of the gripper. With the simulated robot, the fingers are independently actuated, and so the object may drift to one side.

7.3 Tutorial: Robot Teleop

7.3.1 Using the Robot Joystick

Each Fetch and Freight ship with a robot joystick. Whenever the robot drivers are running, so is joystick teleop. The joystick is capable of controlling the movement of the robot base, torso, head and gripper.

Button #	Function (details below)
0	Open gripper
1	Control robot turning
2	Control forward/backward driving
3	Close gripper
4	Disable motor position holding
5	Not used
6	Arm tuck
7	Not used
8	Head control deadman
9	Not used
10	Primary deadman
11	Not used
12	Torso up
13	Not used
14	Torso down
15	Not used
16	Pair/unpair with robot

To pair the controller with the robot, press the middle button (16) once the robot has powered on. The controller will vibrate once successful. To unpair, hold the button for 10 s. The LED indicator on top will turn off.

To drive the robot base, hold the primary deadman button (button 10 above) and use the two joysticks. The left joystick controls turning velocity while the right joystick controls forward velocity.

> **Warning:** Whenever driving the robot, always lower the torso and tuck the arm to avoid potentially unstable operation.

To control the head, release the primary deadman and hold the head deadman (button 8). The left joystick now controls head pan while the right joystick controls head tilt.

To move the torso up, hold the primary deadman and press the triangle button (12). To move the torso down, hold the primary deadman and press the X (14).

To close the gripper, hold the primary deadman and press the close button (3). To open, hold the primary deadman and press the open button (0).

Some controllers, such as the arm and head controllers, will attempt to hold position indefinitely. Sometimes this is not desired. Holding button (4) for 1 second will stop all controllers except the base controller and the arm gravity compensation.

7.3.2 Moving the Base with your Keyboard

Note: You will need a computer with ROS installed to properly communicate with the robot. Please consult the ROS Wiki for more information. We strongly suggest an Ubuntu machine with ROS Indigo installed.

To teleoperate the robot base in simulation, we recommend using the `teleop_twist_keyboard.py` script from teleop_twist_keyboard package.

```
>$ export ROS_MASTER_URI=http://<robot_name_or_ip>:11311
>$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

7.3.3 Software Runstop

In addition to the runstop button on the side of the robot, similar software functionality is also available, allowing for button presses on the PS3 controller or a program to disable the breakers. This functionality is available in release 0.7.3 of the fetch_bringup package. The teleop portion is disabled by default.

Using Software Runstop

To activate the software runstop, publish True to the /enable_software_runstop topic.

Alternately, with the teleop runstop enabled, pressing both of the right trigger buttons (buttons 9 and 11) will activate the software runstop. The software_runstop.py script in fetch_bringup can be modified to change the button(s) for the software runstop.

Once activated, the software runstop can be deactivated by (1) toggling the hardware runstop, or (2) disabling the software runstop by passing False to the /enable_software_runstop topic.

Enable Teleop Software Runstop

Note: In order to edit the robot.launch file, you will need to use a terminal editor (such as nano or vim), or use the -X flag with SSH to use a graphical editor (such as gedit). Additionally, the editor must be launched with *sudo*. Instructions below use nano.

To enable the software runstop, first SSH into the robot, and then modify the robot drivers launch file to use it.

We need to modify the robot.launch file to pass the correct arg to the software runstop script:

```
>$ sudo nano /etc/ros/indigo/robot.launch
```

In this file there should be a Software Runstop entry near the end. By default this entry contains an args line, with a value of "-a -b -g". To add teleop control, add the "-t" flag as well. This section will then look like the below. If your robot is an older one and does not have a Software Runstop entry, you will want to simply copy the block the below.

```
<!-- Software Runstop -->
<include file="$(find fetch_bringup)/launch/include/runstop.launch.xml">
  <arg name="flags" value="-a -b -g -t" />
</include>
```

Note that the -a, -b, -g flags correspond to letting the software runstop control the *arm, base and gripper breakers*, respectively.

Additionally, if completely disabling the software runstop functionality is desired, the above section in robot.launch can be commented out or removed.

Finally, restart the drivers so that our changes take effect:

```
>$ sudo service robot stop && sudo service robot start
```

7.4 Tutorial: Navigation

Once you have Fetch or Freight running, you can start navigating. Fetch and Freight ship with configurations for using the ROS Navigation Stack. A number of tutorials related to navigation can be found in the documentation on the ROS Wiki.

7.4.1 Running Navigation in Gazebo Simulation

To run navigation in simulation, launch the navigation launch file from the `fetch_gazebo_demo` package:

```
>$ roslaunch fetch_gazebo_demo fetch_nav.launch
```

7.4.2 Running Navigation on a Real Robot

When running navigation on a robot, first you will need to build a map, See the next section for a how-to. Then you will need to supply the map to the navigation launch file from the `fetch_navigation` package:

```
>$ roslaunch fetch_navigation fetch_nav.launch map_file:=/path/to/map.yaml
```

7.4.3 Building A Map

The launch file for navigation in Gazebo depends on a pre-built map of the environment. In order to use navigation in the real world, you will need to first build a map of your environment:

```
>$ roslaunch fetch_navigation build_map.launch
```

Once you launch build_map, you will want to tele-operate the robot the robot around and build the map, which can be visualized in RVIZ.

Note: The build_map.launch file is not intended to be run at the same time as fetch_nav.launch

---

While driving the robot around, you can view the map in RVIZ. Once you are happy with the map, you can save the map:

```
>$ rosrun map_server map_saver -f <map_directory/map_name>
```

The map saver will create two files in the specified `map_directory`. The directory must already exist. The two files are map_name.pgm and map_name.yaml. The first is the map in a `.pgm` image format, and the second is a YAML file that specifies metadata for the image. These files can then be served by the map_server:

```
>$ rosrun map_server map_server <map.yaml>
```

The fetch_nav.launch file used above launches an instance of map_server. It has three arguments which control the behavior:

Argument	Meaning
map_file	YAML file containing map metadata
map_keepout_file	Additional YAML file containing metadata for a keepout map
use_keepout	Whether to load and use a keepout map (default: False)

You can either pass the arguments from the command line, like:

```
>$ roslaunch fetch_navigation fetch_nav.launch map_file:=/path/to/map.yaml
```

Or create a new launch file in your own package which includes launch file and passes in arguments:

```
<launch>
  <include file="$(find fetch_navigation)/launch/fetch_nav.launch" >
    <arg name="map_file" value="$(find my_package)/maps/my_map.yaml" />
    <arg name="map_keepout_file" value="$(find my_package)/maps/my_keepout_map.yaml" />
    <arg name="use_keepout" value="true" />
  </include>
</launch>
```

The "keepout" map can be created by copying the YAML file of your saved map, editing the name of the `.pgm` file and then copying the `.pgm` file. You can then open the `.pgm` file in an image editor, such as GIMP, and black out areas that you do not want the robot to drive through. This must be done in a separate map that is only used for planning so that the edits do not disturb the functionality of localization (AMCL).

7.4.4 Sending Waypoints

The easiest way to send a goal to the navigation stack is using RVIZ and the `2D Nav Goal` button. See the tutorial on using RVIZ with navigation in the RVIZ documentation

However, you probably want to program your robot. There is a tutorial on commanding the robot with C++. For examples in Python, look at the demo.py code in the `fetch_gazebo_demo` package.

7.5 Tutorial: Manipulation

Once you have Fetch running, you can start moving the arm with MoveIt!

7.5.1 Getting Started Demo

The easiest way to run MoveIt! is to run the demo launch file, which does not require any simulator or robot and brings up a fully configured RVIZ instance:

```
>$ roslaunch fetch_moveit_config demo.launch
```

Within this demo you can use the sliders of the joint state publisher window to move the joints to new starting positions and use interactive markers to create new locations to plan to and from.

7.5.2 Running the Pick and Place Demo

See *Running the Mobile Manipulation Demo*.

7.5.3 Running MoveIt! on a Robot

To run MoveIt! on a real or simulated robot, launch the move_group.launch file from the `fetch_moveit_config` package:

```
>$ roslaunch fetch_moveit_config move_group.launch
```

Once launched you can send commands to move the arm using the MoveIt! Rviz Plugin or use the programming interface, `move_group_interface`, in either C++ or Python.

7.5.4 Simple MoveIt! Disco Example

This python script will run the robot through a simple disco dance motion.

```
#!/usr/bin/env python

# simple_disco.py: Move the fetch arm through a simple disco motion
import rospy
from moveit_msgs.msg import MoveItErrorCodes
```

```
from moveit_python import MoveGroupInterface, PlanningSceneInterface

# Note: fetch_moveit_config move_group.launch must be running
# Safety!: Do NOT run this script near people or objects.
# Safety!: There is NO perception.
#          The ONLY objects the collision detection software is aware
#          of are itself & the floor.
if __name__ == '__main__':
    rospy.init_node("simple_disco")

    # Create move group interface for a fetch robot
    move_group = MoveGroupInterface("arm_with_torso", "base_link")

    # Define ground plane
    # This creates objects in the planning scene that mimic the ground
    # If these were not in place gripper could hit the ground
    planning_scene = PlanningSceneInterface("base_link")
    planning_scene.removeCollisionObject("my_front_ground")
    planning_scene.removeCollisionObject("my_back_ground")
    planning_scene.removeCollisionObject("my_right_ground")
    planning_scene.removeCollisionObject("my_left_ground")
    planning_scene.addCube("my_front_ground", 2, 1.1, 0.0, -1.0)
    planning_scene.addCube("my_back_ground", 2, -1.2, 0.0, -1.0)
    planning_scene.addCube("my_left_ground", 2, 0.0, 1.2, -1.0)
    planning_scene.addCube("my_right_ground", 2, 0.0, -1.2, -1.0)

    # TF joint names
    joint_names = ["torso_lift_joint", "shoulder_pan_joint",
                   "shoulder_lift_joint", "upperarm_roll_joint",
                   "elbow_flex_joint", "forearm_roll_joint",
                   "wrist_flex_joint", "wrist_roll_joint"]
    # Lists of joint angles in the same order as in joint_names
    disco_poses = [[0.0, 1.5, -0.6, 3.0, 1.0, 3.0, 1.0, 3.0],
                   [0.133, 0.8, 0.75, 0.0, -2.0, 0.0, 2.0, 0.0],
                   [0.266, -0.8, 0.0, 0.0, 2.0, 0.0, -2.0, 0.0],
                   [0.385, -1.5, 1.1, -3.0, -0.5, -3.0, -1.0, -3.0],
                   [0.266, -0.8, 0.0, 0.0, 2.0, 0.0, -2.0, 0.0],
                   [0.133, 0.8, 0.75, 0.0, -2.0, 0.0, 2.0, 0.0],
                   [0.0, 1.5, -0.6, 3.0, 1.0, 3.0, 1.0, 3.0]]

    for pose in disco_poses:
        if rospy.is_shutdown():
            break

        # Plans the joints in joint_names to angles in pose
        move_group.moveToJointPosition(joint_names, pose, wait=False)

        # Since we passed in wait=False above we need to wait here
        move_group.get_move_action().wait_for_result()
        result = move_group.get_move_action().get_result()

        if result:
            # Checking the MoveItErrorCode
            if result.error_code.val == MoveItErrorCodes.SUCCESS:
                rospy.loginfo("Disco!")
            else:
                # If you get to this point please search for:
                # moveit_msgs/MoveItErrorCodes.msg
```

```
                rospy.logerr("Arm goal in state: %s",
                             move_group.get_move_action().get_state())
        else:
            rospy.logerr("MoveIt! failure no result returned.")

    # This stops all arm movement goals
    # It should be called when a program is exiting so movement stops
    move_group.get_move_action().cancel_all_goals()
```

7.5.5 Simple MoveIt! Wave Example

This python script will cause the robot to do a simple "wave-like" motion until the script is stopped with ctrl-c

```
#!/usr/bin/env python

# wave.py: "Wave" the fetch gripper
import rospy
from moveit_msgs.msg import MoveItErrorCodes
from moveit_python import MoveGroupInterface, PlanningSceneInterface
from geometry_msgs.msg import PoseStamped, Pose, Point, Quaternion

# Note: fetch_moveit_config move_group.launch must be running
# Safety!: Do NOT run this script near people or objects.
# Safety!: There is NO perception.
#          The ONLY objects the collision detection software is aware
#          of are itself & the floor.
if __name__ == '__main__':
    rospy.init_node("hi")

    # Create move group interface for a fetch robot
    move_group = MoveGroupInterface("arm_with_torso", "base_link")

    # Define ground plane
    # This creates objects in the planning scene that mimic the ground
    # If these were not in place gripper could hit the ground
    planning_scene = PlanningSceneInterface("base_link")
    planning_scene.removeCollisionObject("my_front_ground")
    planning_scene.removeCollisionObject("my_back_ground")
    planning_scene.removeCollisionObject("my_right_ground")
    planning_scene.removeCollisionObject("my_left_ground")
    planning_scene.addCube("my_front_ground", 2, 1.1, 0.0, -1.0)
    planning_scene.addCube("my_back_ground", 2, -1.2, 0.0, -1.0)
    planning_scene.addCube("my_left_ground", 2, 0.0, 1.2, -1.0)
    planning_scene.addCube("my_right_ground", 2, 0.0, -1.2, -1.0)

    # This is the wrist link not the gripper itself
    gripper_frame = 'wrist_roll_link'
    # Position and rotation of two "wave end poses"
    gripper_poses = [Pose(Point(0.042, 0.384, 1.826),
                          Quaternion(0.173, -0.693, -0.242, 0.657)),
                     Pose(Point(0.047, 0.545, 1.822),
                          Quaternion(-0.274, -0.701, 0.173, 0.635))]

    # Construct a "pose_stamped" message as required by moveToPose
    gripper_pose_stamped = PoseStamped()
    gripper_pose_stamped.header.frame_id = 'base_link'
```

```
    while not rospy.is_shutdown():
        for pose in gripper_poses:
            # Finish building the Pose_stamped message
            # If the message stamp is not current it could be ignored
            gripper_pose_stamped.header.stamp = rospy.Time.now()
            # Set the message pose
            gripper_pose_stamped.pose = pose

            # Move gripper frame to the pose specified
            move_group.moveToPose(gripper_pose_stamped, gripper_frame)
            result = move_group.get_move_action().get_result()

            if result:
                # Checking the MoveItErrorCode
                if result.error_code.val == MoveItErrorCodes.SUCCESS:
                    rospy.loginfo("Hello there!")
                else:
                    # If you get to this point please search for:
                    # moveit_msgs/MoveItErrorCodes.msg
                    rospy.logerr("Arm goal in state: %s",
                                 move_group.get_move_action().get_state())
            else:
                rospy.logerr("MoveIt! failure no result returned.")

    # This stops all arm movement goals
    # It should be called when a program is exiting so movement stops
    move_group.get_move_action().cancel_all_goals()
```

7.5.6 More information and Tutorials on MoveIt!

General information, Documentation and Tutorials available at moveit.ros.org

7.6 Tutorial: Perception

7.6.1 Head Camera Topics

The API for the head camera is documented under *Head Camera Interface*.

Some resources for accessing and processing camera data are:

- OpenCV is a generic computer vision library which has good support within ROS.
- Point Cloud Library allows manipulation of 3-dimensional images, or point clouds.
- cv_bridge is a ROS package that allows converting ROS image messages into OpenCV data structures in C++ or Python.
- pcl_conversions is a ROS package for converting between ROS PointCloud2 messages and PCL data types in C++.
- pcl_ros is a ROS package that contains several nodelets for commonly used PCL components such as voxel grid filters for downsampling a point cloud or pass through filters for filtering out data beyond a certain distance.

7.6.2 Running the Pick and Place Demo

See *Running the Mobile Manipulation Demo*.

7.7 Tutorial: Auto Docking

Fetch Robotics has released a ROS package for automatically docking your robot with a Fetch charging dock. This package uses the scanning laser range finder to detect the profile of the charge dock and steers the robot onto the dock. Since the profile of the charge dock must be clearly visible, it is important that there is separation between docks and any sort of laser-height obstacle on either side of the dock.

Warning: Charge dock installation must follow the specifications!

7.7.1 Installing fetch_auto_dock

To install the package:

```
sudo apt-get update
sudo apt-get install ros-indigo-fetch-auto-dock
```

7.7.2 Running the Docking Node

To start the auto docking node in a demonstration mode:

```
roslaunch fetch_auto_dock auto_dock.launch
```

This will bring up three ROS nodes. The first ROS node is the auto docking action server. This node can be asked to dock with a charging dock through a ROS action interface. A second and third nodes monitor the PS3 joystick and can trigger docking and undocking as described below:

- To dock your robot, point the robot at the charge dock and press the "circle" button on the PS3 controller. Docking works best when the robot is about 0.75-1.0 meters away from the dock and pointed directly at it, however docking should work from a variety of angles.
- To undock your robot when it is on a charge dock, press the "square" button on the PS3 joystick and the robot will back off the dock and then turn in place so it is pointed away from the charge dock.

Warning: The docking controller does not have collision avoidance. Do not leave obstacles on the charge dock.

7.7.3 Auto Docking Programmatically

The auto dock node exposes a ROS action interface which can be used to dock the robot. This action interface is also used by the demonstration python script.

This action is in the *dock* namespace and accepts a fetch_auto_dock_msgs/Dock action message.

There are several fields in the goal:

- `dock_pose` is a geometry_msgs/PoseStamped message which specifies where the dock center is located. This can be in any valid frame as the action server node will use TF to transform the pose into the odometry frame.
- `use_move_base` is not currently implemented.

The result includes a single boolean `docked` which tells whether the robot has actually docked and is charging.

The following Python code will dock the robot with a dock that is 1.0 meters in front of it:

```
#!/usr/bin/env python
import rospy
import actionlib
from fetch_auto_dock_msgs.msg import DockAction, DockGoal

# Create a ROS node
rospy.init_node("dock_the_robot")

# Create an action client
client = actionlib.SimpleActionClient("/dock", DockAction)
client.wait_for_server()

# Create and send a goal
goal = DockGoal()
goal.dock_pose.header.frame_id = "base_link"
goal.dock_pose.pose.position.x = 1.0
goal.dock_pose.pose.orientation.z = 1.0
client.send_goal(goal)
```

The fetch_auto_dock node also provides an `undock` action interface which can be used to undock the robot from the charge dock. The goal to undock has a single field

- `rotate_in_place` if set to true, the robot will back off the dock and then rotate 180 degrees so it is facing off the dock. This can be very useful since navigation probably cannot get the robot off the dock when it is facing at the dock.

7.8 Tutorial: Calibrating Fetch

Calibrating your Fetch Research Edition Robot is essential to getting the best performance. Fetch Robotics recommends re-calibrating the robot after every shipping or travel excursion your robot undertakes.

7.8.1 Understanding Calibration

The *fetch_calibration* package provides a method of making sure that the hand-eye coordination of the robot is well calibrated. Calibration involves:

- Moving the arm to a series of poses
- At each pose, recording the joint_angles reported by the drivers. Calibration then blinks the gripper LEDs several times to find their pose in the camera frame and records it.
- Performing a non-linear optimization which adjusts joint offsets and the head camera pose to minimize the error between the expected pose of the the LEDs based on projection of the kinematics of the arm and the actual pose seen by the camera.
- Updating the URDF and robot launch files with newly determined offsets.

The *Upstart Services* that start the robot will use the launch file in */etc/ros/indigo/robot.launch*. Therefore, the last step in calibration is to update that launch file and restart the drivers. Currently, the following aspects are updated:

- The URDF file is copied to /etc/ros/indigo, and the name of the file is put into robot.launch. The calibration offsets are updated in the URDF file.

- The camera calibration YAML files are copied to /etc/ros/indigo, and the name of the files are put into robot.launch.
- The head_camera driver has two parameters, z_offset_mm and z_scale which are calibrated. Their updated values are stored in robot.launch.

7.8.2 Calibrate Robot Tool

The *fetch_calibration* package provides a tool called *calibrate_robot* which fully automates tasks related to calibration. Once you have sourced the ROS setup.bash, running *calibrate_robot* without any arguments will show the list of valid arguments:

```
>$ calibrate_robot
usage: calibrate_robot [-h] [--arm] [--base] [--install] [--reset] [--restore]
                       [--date] [--directory DIRECTORY]

Calibrate the robot, update files in /etc/ros

optional arguments:
  -h, --help            show this help message and exit
  --arm                 Capture arm/head calibration data
  --base                Capture base calibration data
  --install             Install new calibration to /etc/ros (restarts drivers)
  --reset               Reset the calibration to factory defaults (restarts
                        drivers)
  --restore             Restore the previous calibration
  --date                Get the timestamp of the current calibration
  --directory DIRECTORY
                        Directory to load calibration from or backup to
```

A useful command to note is the *–date* option, which will return the date that the current calibration was generated (or 'uncalibrated' if the robot has not yet been calibrated):

```
>$ calibrate_robot --date
2015-06-06 23:20:18
```

7.8.3 Running Calibration

Warning: Calibration will cause the arm to move through the environment. Raise the torso to full extension and be sure that the robot has at least 1 meter of free space all around it.

The first step to calibrate the robot is to reset the calibration to factory defaults:

```
>$ calibrate_robot --reset
```

This command might ask for your password, as it requires sudo to update the files in /etc/ros/. This will also restart the drivers to make the changes take effect.

Warning: When calibrating, make sure that no other applications are subscribed to the head_camera topics. Other applications, even RVIZ, connecting to the head_camera driver between the restart of drivers and the start of calibration may adversely affect the auto_exposure settings of the camera.

The robot is now ready to calibrate. The following command will move the arm through a series of poses and upon completion of the calibration will update the robot configuration in /etc/ros/indigo:

```
>$ calibrate_robot --arm --install
```

Finally, after the new calibration has been installed into /etc/ros/indigo, you can view the calibration by opening RVIZ and seeing that the robot model overlaps with the point cloud from the head camera. If for some reason, the calibration is not good, you can restart from the *–reset* command or roll back to the previous calibration with:

```
>$ calibrate_robot --restore
```

7.8.4 Calibrating Fetch Torso

When the torso controller board is first powered, it uses measurements from two different position sensors to determine the absolute starting position of the torso. Once the absolute starting position of the torso is determined, the position measurement will retain millimeter precision as long torso remains powered.

To work properly, the two torso sensors must be calibrated together. If the sensors are not properly calibrated, the calculation of the initial torso position could be incorrect in some situations. This problem is more likely to occur if the torso is first powered when in the "up" position. A bad torso position should be easy to detect when using RVIZ since the torso position shown in RVIZ will always be more than +/-10cm different than the true torso position.

The torso sensors are calibrated in production, so they will not usually need to be recalibrated. If there seems to be a torso positioning problem, the torso calibration tool should be first used to verify the calibration of the torso.

In release 0.7.4 of fetch_drivers package there is a tool to verify or calibrate the torso sensors. The tool has two options: verify and calibrate. The verify option will only verify that the calibration is good, it will not change any stored calibration parameters. The calibrate option will calibrate the sensors and update the parameters stored on the torso controller.

For both options, the torso will travel through its entire range of motion while sensor data is collected. While the tool is being run, the robot drivers will be stopped and the robot arm will not hold its position. Because of this, the arm should be tucked or soft fabric or cardboard should be placed between arm and base to avoid scratching any covers.

Torso Calibration Procedure

Warning: During torso calibration the arm will not hold position. Place cardboard or soft fabric between arm and base to avoid scratching covers during data collection.

Follow these steps in order to verify or calibrate the torso position sensors:

1. Move torso to lowest position, and tuck the arm.
2. Place a protective barrier between the arm and top base cover.
3. Disable robot drivers by running : `sudo service robot stop`
4. Run torso calibration tool:

- To run calibration : `rosrun fetch_drivers torso_calibrate calibrate`
- To verify calibration : `rosrun fetch_drivers torso_calibrate verify`

6. Wait for torso to collect sensor data. The torso will move upwards in small increments through the entire range of motion. A clicking sound will be produced by the torso while moving, and is normal.
7. Cycle Run-stop (optional). Sometimes tool will request that run-stop be cycled after it completes. Cycling run-stop will cycle power to the torso controller board, and is required in some situations.
8. On tool has completed, restart robot drivers with `sudo service robot start`

Verify Output

Verification will produce output stating whether sensors are well calibrated. If the sensor calibration is good the tool will output something similar to:

> VERIFY PASSED : max sensor error of 0.0109411 is within acceptable limit

Otherwise it will produce output like:

> VERIFY FAILED : max sensor error of 0.0501323 is larger than acceptable limit of 0.04

When verification fails, run calibration produce.

Note: The value for max sensor error is the mismatch between the two torso sensors. The accuracy of the torso position measurement is unrelated to this value.

Calibrate Output

Once the tool has completed the calibration procedure it will check the expected results of calibration. If everything checks out, the tool will output something similar to:

```
VERIFY PASSED : max sensor error of 0.0116824 is within acceptable limit
```

If there was a problem calculating good calibration parameters, the output might look like:

```
VERIFY FAILED : max sensor error of 0.0501323 is larger than acceptable limit of 0.04
```

In case of failure, the torso sensor may be malfunctioning or damaged and a support ticket should be created.

7.9 Tutorial: Fetch Programming by Demonstration

The fetch_pbd package is based on PR2 Programming by Demonstration. This version is for Fetch. It will not work on Freight because Freight does not have an arm.

The original PR2 Programming by Demonstration was done by Maya Cakmak and the Human-Centered Robotics Lab at the University of Washington.

7.9.1 System Requirements

This PbD is designed for Ubuntu 14.04 and ROS Indigo.

7.9.2 Installing from Source

Clone the repository and build on the robot:

```
>$ cd ~/catkin_ws/src
>$ git clone https://github.com/fetchrobotics/fetch_pbd.git
>$ cd ~/catkin_ws
>$ catkin_make
```

7.9.3 Running

Run these commands on a terminal on the Fetch:

```
>$ source ~/catkin_ws/devel/setup.bash
>$ roslaunch fetch_pbd_interaction pbd.launch
```

You can run the backend without the "social gaze" head movements or without the sounds by passing arguments to the launch file:

```
>$ source ~/catkin_ws/devel/setup.bash
>$ roslaunch fetch_pbd_interaction pbd.launch social_gaze:=false play_sound:=false
```

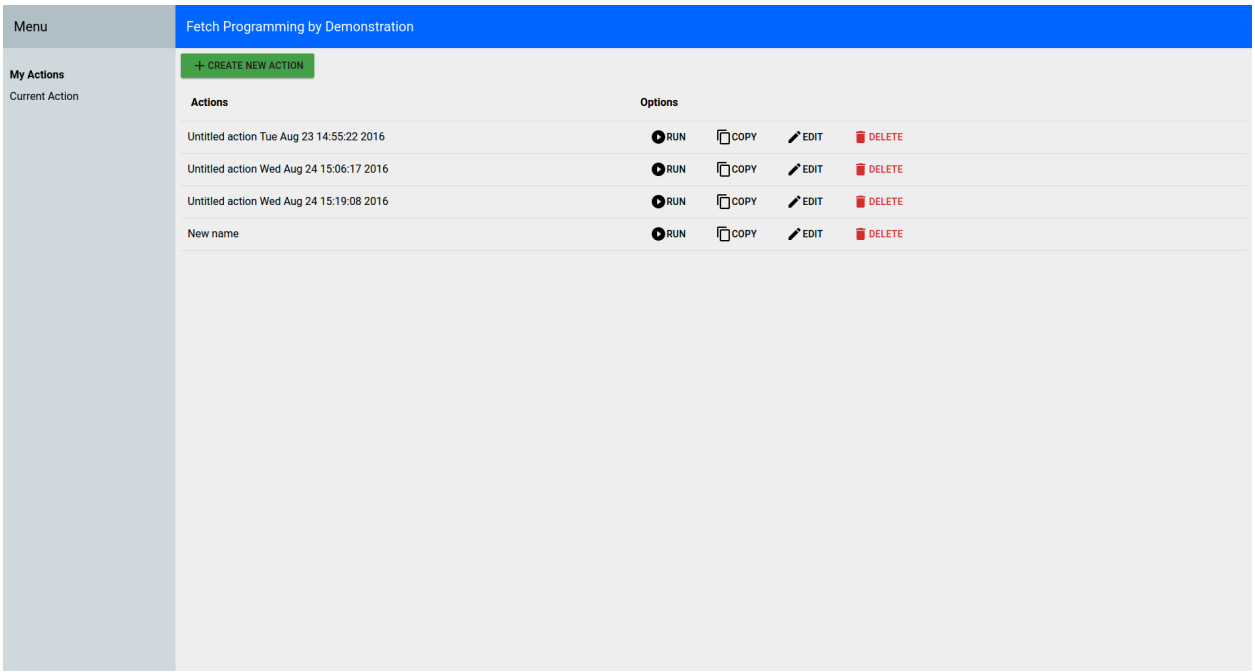
You can also pass arguments to the launch file to save your actions to a json file or load them from a json file. This behaviour is a bit complicated. It is recommended that you specify the full path to files or else it will look in your .ros folder. If you specify a from_file then actions will be loaded from that file. They will replace the ones in your session database. Whatever was in your session database will get stored in a timestamped file in your .ros folder (not overwritten). If you specify a to_file then whatever is in your current session file be saved to that file.

```
>$ source ~/catkin_ws/devel/setup.bash
>$ roslaunch fetch_pbd_interaction pbd.launch from_file:=/full/path/from.json to_file:=/full/path/to
```

7.9.4 Using the GUI

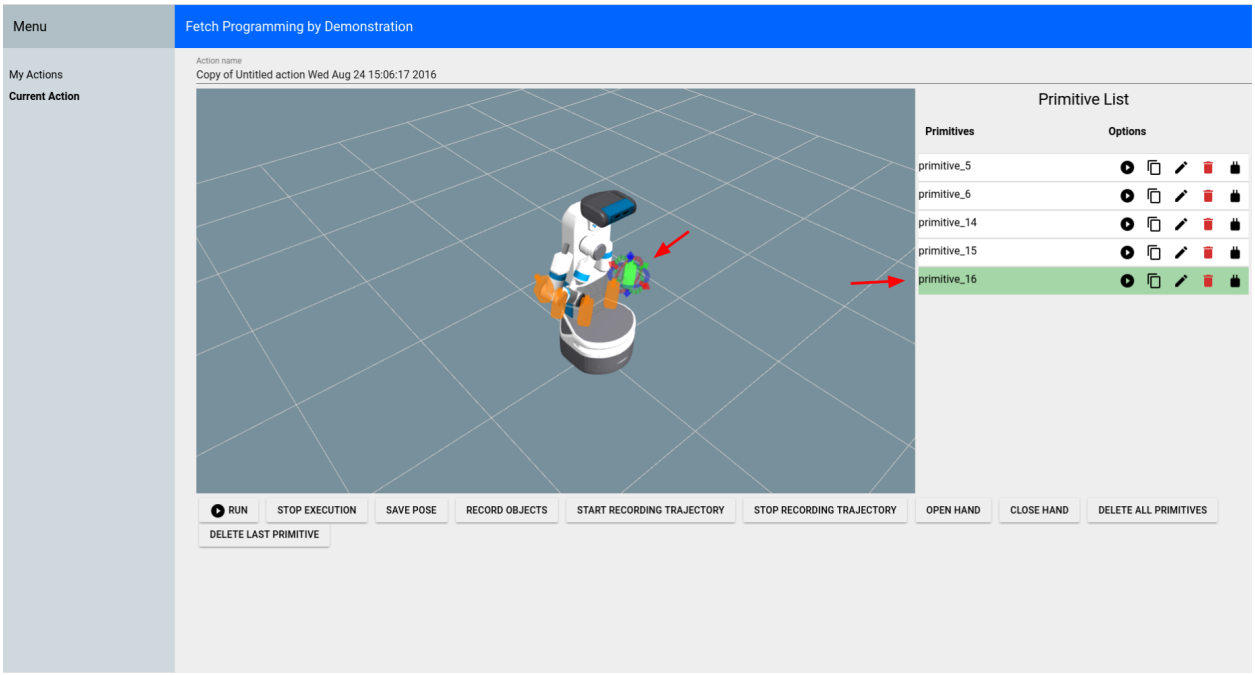
In your browser go to ROBOT_HOSTNAME:8080 in your browser to use the GUI. This can be used on mobile as well. The mobile version does not show the visualizer, but can be useful for saving/deleting

The main page lists all the available actions.



You can directly run/copy/delete actions from the main page. Or hit the "Edit" button to see more information on that action.

On the "Current Action" screen, most of the buttons are pretty self-explanatory. You can execute the entire action using the "Run" button at the bottom of the screen. This will execute all of the primitives in the order they appear in the Primitive List. You can click on a specific primitive (either the marker or the list item), to highlight the primitive.



You can show/hide the markers for each primitive by clicking the marker icon for the primitive in the Primitive List.

You can change the order of the primitives by dragging them to a new position in the list.

You can edit the position and orientation of certain primitives by clicking the edit icon or by moving the interactive marker.

You can change the frame that certain primitives are relative to by right-clicking the marker.

You can also change the name of the action.

7.9.5 Code Interface

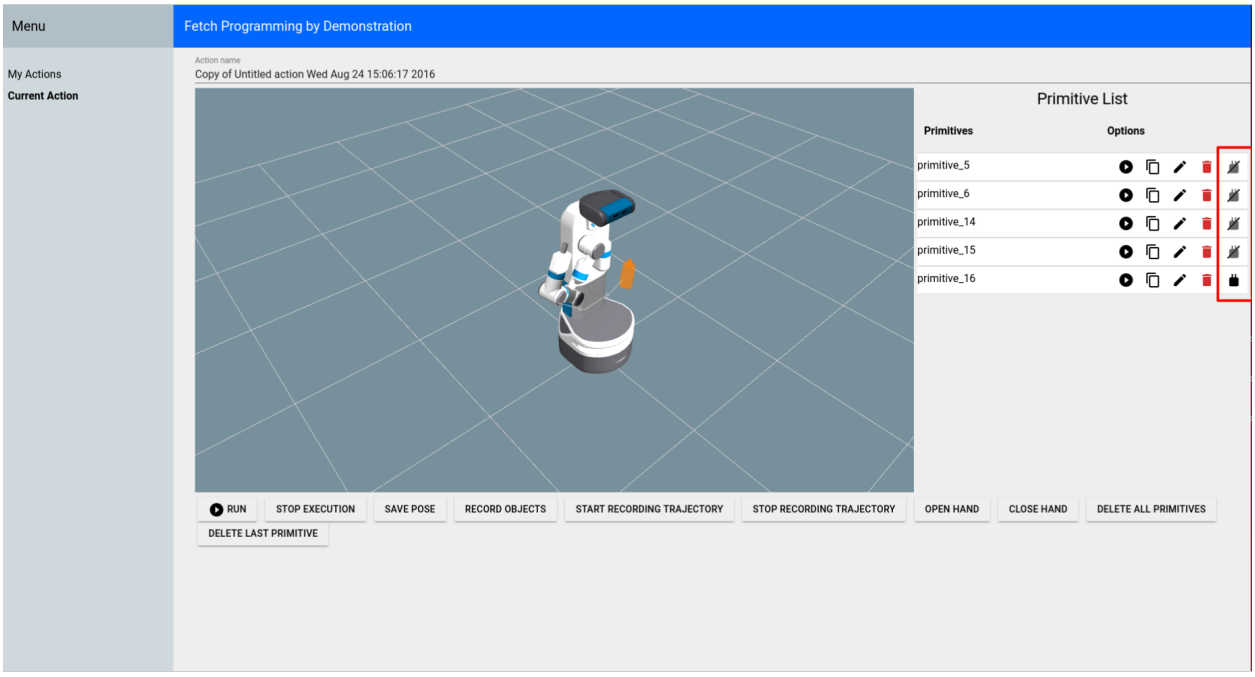
You can also access the actions you've programmed through code. You still need to run pbd_backend.launch.

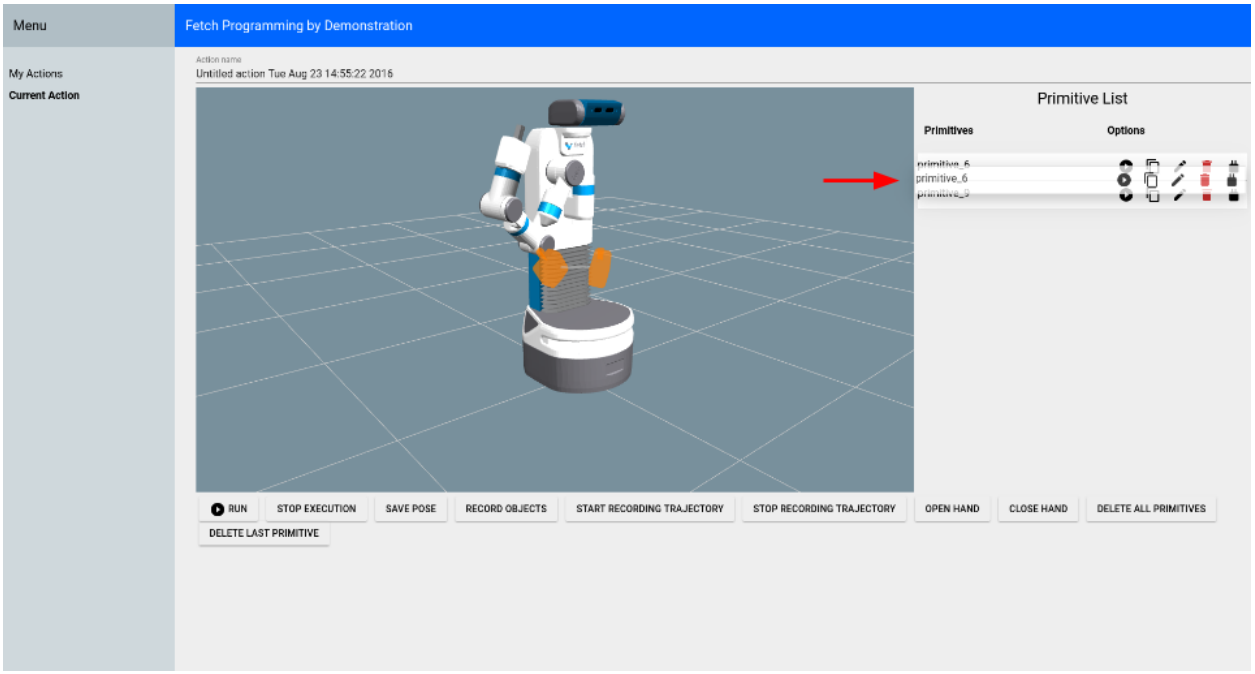
```
>$ source ~/catkin_ws/devel/setup.bash
>$ rosrun fetch_pbd_interaction demo.py
```

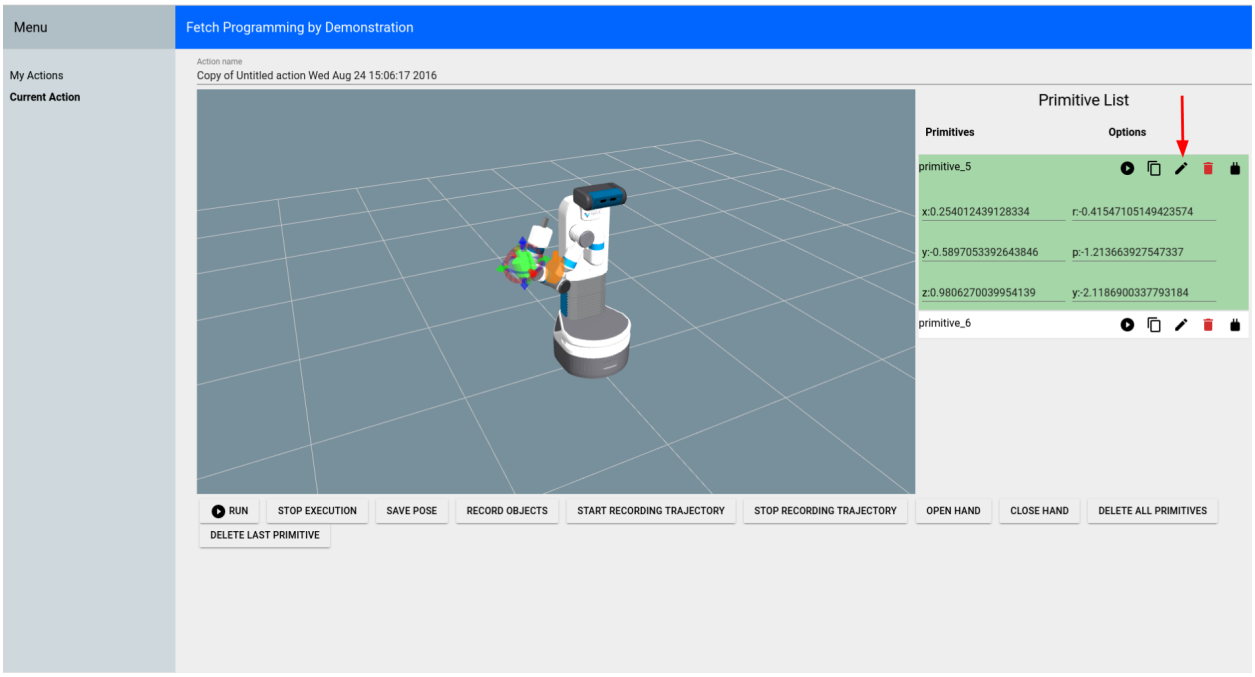
7.9.6 System Overview

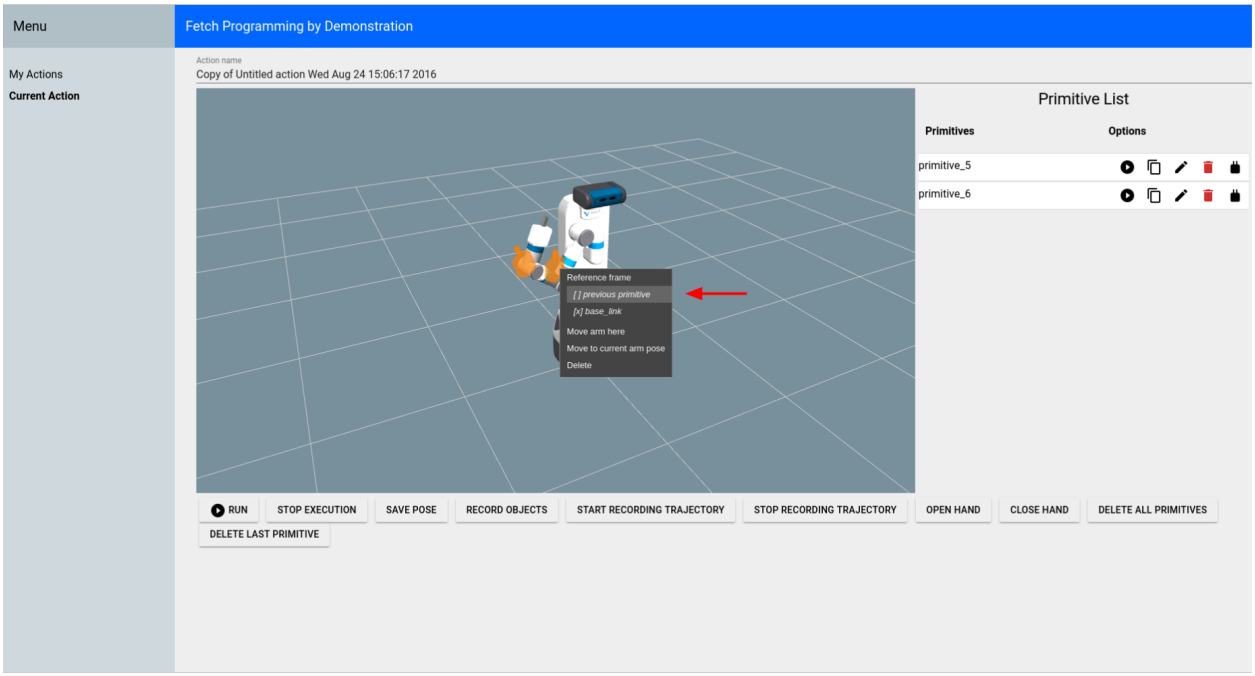
Interaction Node: The pbd_interaction_node.py handles the interaction between speech/GUI and the rest of the system. Changes happen through the update loop in interaction.py and also through the callbacks from speech/GUI commands. interaction.py also subscribes to updates from the pbd_world_node.py, which notifies it of changes in objects in the world. Through callbacks and the update loop, interaction.py hooks in to session.py. session.py handles creating actions and primitives and saving them to the database.

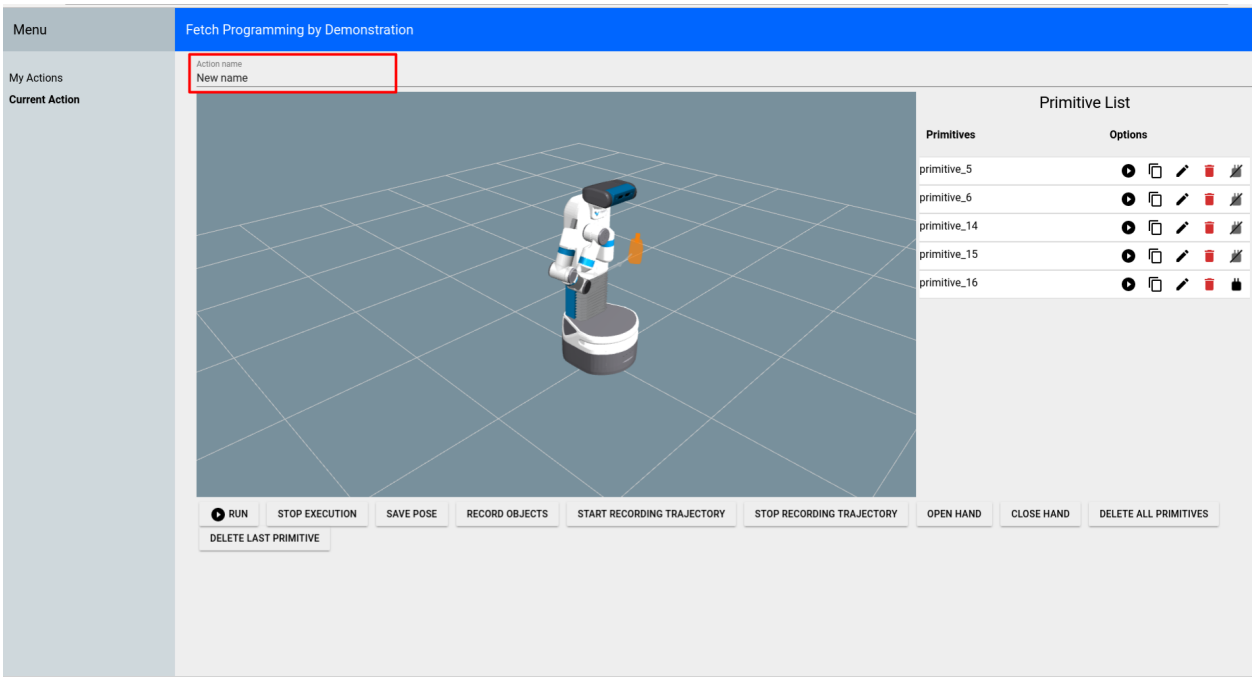
Arm Control Node: The pbd_arm_control_node.py is how the robot's arm is controlled to execute actions/primitives. It provides a lower level service interface to move the arm. The interaction node interacts with this through the interface in robot.py.


Menu
Fetch Programming by Demonstration
My Actions
Current Action
Action name
Copy of Untitled action Wed Aug 24 15:06:17 2016
Primitive List
Primitives
Options
primitive_5
primitive_6
primitive_14
primitive_15
primitive_16
RUN
STOP EXECUTION
SAVE POSE
RECORD OBJECTS
START RECORDING TRAJECTORY
STOP RECORDING TRAJECTORY
OPEN HAND
CLOSE HAND
DELETE ALL PRIMITIVES
DELETE LAST PRIMITIVE


Menu
Fetch Programming by Demonstration
My Actions
Current Action
Action name
Untitled action Tue Aug 23 14:55:22 2016
Primitive List
Primitives
Options
primitive_6
RUN
STOP EXECUTION
SAVE POSE
RECORD OBJECTS
START RECORDING TRAJECTORY
STOP RECORDING TRAJECTORY
OPEN HAND
CLOSE HAND
DELETE ALL PRIMITIVES
DELETE LAST PRIMITIVE


Menu
Fetch Programming by Demonstration
My Actions
Current Action
Action name
Copy of Untitled action Wed Aug 24 15:06:17 2016
Primitive List
Primitives
Options
primitive_5
x:0.254012439128334
r:-0.41547105149423574
y:-0.5897053392643846
p:-1.213663927547337
z:0.9806270039954139
y:-2.1186900337793184
primitive_6
RUN
STOP EXECUTION
SAVE POSE
RECORD OBJECTS
START RECORDING TRAJECTORY
STOP RECORDING TRAJECTORY
OPEN HAND
CLOSE HAND
DELETE ALL PRIMITIVES
DELETE LAST PRIMITIVE


Menu
Fetch Programming by Demonstration
My Actions
Current Action
Action name
Copy of Untitled action Wed Aug 24 15:06:17 2016
Primitive List
Primitives
Options
primitive_5
primitive_6
Reference frame
[ ] previous primitive
[x] base_link
Move arm here
Move to current arm pose
Delete
RUN
STOP EXECUTION
SAVE POSE
RECORD OBJECTS
START RECORDING TRAJECTORY
STOP RECORDING TRAJECTORY
OPEN HAND
CLOSE HAND
DELETE ALL PRIMITIVES
DELETE LAST PRIMITIVE



World Node: The pbd_world_node.py handles the robot's perception of the world. Other nodes ask the world node about the state of the world and can both send and subscribe to updates to the world. Its main function is to provide a list of objects currently in the scene.

Social Gaze Node: The social_gaze_server.py handles the movements of the robot's head. This is also controlled through the robot.py interface. The sounds are also provided through this interface.

CHAPTER

EIGHT

OTHER

8.1 API Overview

Fetch Robotics is committed to providing an exceptional out-of-the-box experience for new robot owners. One of the ways in which we create a great experience is by using standard ROS interfaces. This means that code you might have written for other robots in the past should be easily portable to your new robot.

Whenever possible, we have conformed to the ROS Enhancement Proposals (REPs). These documents provide the foundation of standard ROS interfaces. In addition to REP-compatible interfaces, we have adopted a number of the community-accepted standard interfaces, such as those provided by the control_msgs package.

8.1.1 Arm and Torso

The arm and torso of the robot are controlled by control_msgs/FollowJointTrajectory actions. This action interface is the output of packages such as MoveIt, and allows the arm to execute a pre-defined trajectory. Three interfaces are provided:

- *arm_controller/follow_joint_trajectory* to control just the seven joints of the arm.
- *arm_with_torso_controller/follow_joint_trajectory* to control the seven joints of the arm plus the torso.
- *torso_controller/follow_joint_trajectory* to control just the torso.

Only one controller is allowed to control a joint at a time.

Warning: Fetch is tuned to work with smooth paths. The tuning performs best when paths are properly smoothed and timed, such as those generated from MoveIt! Fetch has a number of mechanisms in place to prevent damage from improper trajectories. Improperly constructed paths may cause motor or breaker shutdown.

In addition to the trajectory controllers, the arm is always running a gravity compensation controller.

8.1.2 Base Interface

Support for mobile bases is quite standard and robust in ROS, however it is one of the older interfaces. As such, it is one of the few interfaces which is not action-based.

The mobile base subscribes to *base_controller/command*, and accepts a geometry_msgs/Twist message.

Only two fields are used in the message:

- `linear.x` specifies the robot's forward velocity
- `angular.z` specifies the robot's turning velocity

User applications will typically not connect directly to *base_controller/command*, but rather to *cmd_vel*. A multiplexer is always running between *cmd_vel/teleop* and *cmd_vel*. Whenever the deadman on the robot controller is held, *cmd_vel/teleop* will override *cmd_vel*. The advantage of having your application publish to *cmd_vel* rather than directly to *base_controller/command* is that you can override bad commands by simply pressing the deadman on the robot controller.

The base controller implements a speed reduction when in the proximity of obstacles. This will not entirely stop the robot if it is about to hit something, but will prevent full speed collisions.

8.1.3 Head Interface

The head exposes two potential interfaces. The first is an ActionServer available on *head_controller/follow_joint_trajectory* which follows a joint trajectory as with the arm and torso (described above).

The second interface is unique to the head, and allows the user to easily point the head (and head sensors) at a point of interest. This action-based interface is available on *head_controller/point_head*. It is of type control_msgs/PointHead. Although the interface currently does not support any of the `pointing_axis` or `pointing_frame` fields, it points the *head_tilt_link* (which is very near the camera optical axis) towards the *target* point to achieve a similar effect. A `min_duration` or `max_velocity` can also be specified.

8.1.4 Gripper Interface

gripper_controller/gripper_action exposes a control_msgs/GripperCommand ActionServer. The gripper command takes in `position` and `effort` as parameters. Generally, the gripper is commanded to a fully closed or fully opened position, so `effort` is used to limit the maximum effort. As the gripper never fully reaches the closed position, the grasp strength will be determined by the maximum effort.

8.1.5 Head Camera Interface

The head camera exposes several topics of interest:

- *head_camera/depth_registered/points* is a sensor_msgs/PointCloud2 which has both 3d and color data. It is published at VGA resolution (640x480) at 15Hz.
- *head_camera/depth_downsampled/points* is a sensor_msgs/PointCloud2 which has only 3d data. It is published at QQVGA (160x120) resolution at 15Hz and is intended primarily for use in navigation/moveit for obstacle avoidance.
- *head_camera/depth/image_raw* is a sensor_msgs/Image. This is unit16 depth image (2D) in mm . It is published at VGA resolution (640x480) at 15Hz.
- *head_camera/depth/image* is a sensor_msgs/Image. This is float depth image (2D) in m. It is published at VGA resolution (640x480) at 15Hz.
- *head_camera/rgb/image_raw* is a sensor_msgs/Image. This is just the 2d color data. It is published at VGA resolution (640x480) at 15Hz.

8.1.6 Laser Interface

base_scan is a sensor_msgs/LaserScan message published at 15Hz.

Note: the raw laser information as reported by the laser hardware is published to *base_scan_raw*. The information published to *base_scan* is filtered to remove shadow points.

8.1.7 IMU Interface

imu is a sensor_msgs/Imu message published at 100Hz. This message contains the linear acceleration and rotational velocities as measured by the IMU located in the base of the robot.

On Fetch robots, the gripper IMU publishes to *gripper_imu*. This is also a sensor_msgs/Imu message published at 100Hz.

The IMUs are not present in the simulated robot.

8.1.8 Resetting Breakers

There are 3 breakers governing power on the fetch. One each for the arm, gripper and base. If the motors are commanded to perform beyond their limits they will shut down for safety. To reset them you will either need to toggle the e-stop or use the following service calls:

- $ rosservice call /arm_breaker false && rosservice call /arm_breaker true
- $ rosservice call /base_breaker false && rosservice call /base_breaker true
- $ rosservice call /gripper_breaker false && rosservice call /gripper_breaker true

8.2 Release Notes

The following release notes detail the updates to packages available on http://packages.fetchrobotics.com. For more details on the changes that have occured in an individual package, please see the CHANGELOG within the installed package.

For information on updating your robot to the latest packages, see *Updating Your Robot*.

8.2.1 May 28, 2016

This sync includes new upstream ROS packages. Notably this release includes updates for a udev rule that maps the PS3 controller to /dev/ps3joy, therefore it is important that you also install the latest fetch-system-config or freight-system-config package depending on your robot model. The *Updating Your Robot* instructions have been updated to note that the correct update command is now:

```
sudo apt-get update
sudo apt-get install --only-upgrade ros-indigo-* f.*-system-config
sudo service robot stop
sudo service robot start
```

New drivers improve charge time and performance. A number of improvements have been made to the fetch_depth_layer including properly supporting deactivate/activate when plans are not in progress.

Updated Fetch Packages:

- fetch-system-config: 0.8-0 -> 0.8-4
- ros-indigo-fetch-bringup: 0.6.0-0 -> 0.7.1-0
- ros-indigo-fetch-drivers: 0.7.4-0 -> 0.7.11-0
- ros-indigo-fetch-depth-layer: 0.7.0-0 -> 0.7.5-0
- ros-indigo-fetch-description: 0.7.0-0 -> 0.7.5-0
- ros-indigo-fetch-gazebo: 0.7.0-0 -> 0.7.5-0

- ros-indigo-fetch-gazebo-demo: 0.7.0-0 -> 0.7.5-0
- ros-indigo-fetch-moveit-config: 0.7.0-0 -> 0.7.5-0
- ros-indigo-fetch-navigation: 0.7.0-0 -> 0.7.5-0
- ros-indigo-fetch-teleop: 0.7.0-0 -> 0.7.5-0
- ros-indigo-freight-bringup: 0.6.0-0 -> 0.7.1-0
- ros-indigo-robot-controllers: 0.4.3-0 -> 0.5.0-0

A full list of new upstream packages can be found on the ROS mailing list

8.2.2 January 21, 2016

This sync includes new upstream ROS packages. New drivers include improvements to charge state estimation and a tool for *in-field calibration of the torso*. Auto docking includes several fixes for TF-related errors, as well as a fix for reliability when the odom frame and dock are aligned.

Updated Fetch Packages:

- ros-indigo-fetch-drivers: 0.7.3-0 -> 0.7.4-0
- ros-indigo-fetch-auto-dock: 0.1.0-0 -> 0.2.1-0

A full list of new upstream packages can be found on the ROS mailing list

8.2.3 November 23, 2015

This sync includes new upstream ROS packages as well as minor bug fixes and improvements to drivers. Notably, the deadman must now be held while tucking the arm, this allows a user to stop the arm tucking should the robot collide with an obstacle in the environment.

Of note, this release also fixes several inconsistencies in the wrist_flex range of the robot. If your robot appears to have an overly limited wrist_flex range, we recommend recalibrating the robot from a clean URDF after updating your packages.

Maps have been removed from the fetch_navigation package and moved to their own package, fetch_maps.

Updated Fetch Packages:

- ros-indigo-fetch-drivers: 0.7.1-0 -> 0.7.3-0
- ros-indigo-fetch-depth-layer: 0.6.2-0 -> 0.7.0-0
- ros-indigo-fetch-description: 0.6.2-0 -> 0.7.0-0
- ros-indigo-fetch-gazebo: 0.6.2-0 -> 0.7.0-0
- ros-indigo-fetch-gazebo-demo: 0.6.2-0 -> 0.7.0-0
- ros-indigo-fetch-moveit-config: 0.6.2-0 -> 0.7.0-0
- ros-indigo-fetch-navigation: 0.6.2-0 -> 0.7.0-0
- ros-indigo-fetch-teleop: 0.6.2-0 -> 0.7.0-0

New Fetch Packages:

- ros-indigo-fetch-maps: 0.7.0-0

A full list of new upstream packages can be found on the ROS mailing list

8.2.4 November 12, 2015

This sync includes new upstream ROS packages as well as the first release of auto docking.

Please note that the MD5 checksum for the dock action will have changed with this release.

Updated Fetch Packages:

- ros-indigo-fetch-drivers: 0.6.3-0 -> 0.7.1-0
- ros-indigo-fetch-auto-dock-msgs: 0.5.2-0 -> 0.6.0-0
- ros-indigo-fetch-driver-msgs: 0.5.2-0 -> 0.6.0-0
- ros-indigo-fetch-gazebo: 0.6.1-0 -> 0.6.2-0
- ros-indigo-fetch-gazebo-demo: 0.6.1-0 -> 0.6.2-0

New Fetch Packages:

- ros-indigo-fetch-auto-dock: 0.1.0

A full list of new upstream packages can be found on the ROS mailing list

8.2.5 August 5, 2015

This sync includes new upstream ROS packages as well as minor fixes to the URDF and calibration.

Updated Fetch Packages:

- ros-indigo-fetch-drivers: 0.6.1-0 -> 0.6.3-0
- ros-indigo-fetch-depth-layer: 0.6.1-0 -> 0.6.2-0
- ros-indigo-fetch-description: 0.6.1-0 -> 0.6.2-0
- ros-indigo-fetch-moveit-config: 0.6.1-0 -> 0.6.2-0
- ros-indigo-fetch-navigation: 0.6.1-0 -> 0.6.2-0
- ros-indigo-fetch-teleop: 0.6.1-0 -> 0.6.2-0

A full list of new upstream packages can be found on the ROS mailing list

8.2.6 July 9, 2015

This sync includes new upstream ROS packages as well as tuck arm functionality from the robot joystick. This release also includes charge level estimates for Fetch and Freight robots.

Updated Fetch Packages:

- ros-indigo-fetch-drivers: 0.5.3-0 -> 0.6.1-0
- ros-indigo-fetch-depth-layer: 0.5.13-0 -> 0.6.1-0
- ros-indigo-fetch-description: 0.5.13-0 -> 0.6.1-0
- ros-indigo-fetch-driver-msgs: 0.5.1-0 -> 0.5.2-0
- ros-indigo-fetch-gazebo: 0.5.0-0 -> 0.6.1-0
- ros-indigo-fetch-gazebo-demo: 0.5.0-0 -> 0.6.1-0
- ros-indigo-fetch-moveit-config: 0.5.13-0 -> 0.6.1-0
- ros-indigo-fetch-navigation: 0.5.13-0 -> 0.6.1-0

- ros-indigo-fetch-teleop: 0.5.13-0 -> 0.6.1-0
- ros-indigo-robot-calibration: 0.4.0-0 -> 0.5.2-0
- ros-indigo-robot-calibration-msgs: 0.4.0-0 -> 0.5.2-0

New Fetch Packages:

- ros-indigo-fetch-auto-dock-msgs: 0.5.2-0

A full list of new upstream packages can be found on the ROS mailing list

8.2.7 June 8, 2015

First publicly available release.

New Fetch Packages:

- ros-indigo-fetch-drivers: 0.5.3-0
- ros-indigo-fetch-depth-layer: 0.5.13-0
- ros-indigo-fetch-description: 0.5.13-0
- ros-indigo-fetch-driver-msgs: 0.5.1-0
- ros-indigo-fetch-gazebo: 0.5.0-0
- ros-indigo-fetch-gazebo-demo: 0.5.0-0
- ros-indigo-fetch-moveit-config: 0.5.13-0
- ros-indigo-fetch-navigation: 0.5.13-0
- ros-indigo-fetch-teleop: 0.5.13-0

A full list of new upstream packages can be found on the ROS mailing list

8.3 Frequently Asked Questions

8.3.1 How can I cite Fetch or Freight in an Academic Paper?

We recommend citing our workshop paper on Fetch & Freight:

```
Fetch & Freight: Standard Platforms for Service Robot Applications
Melonee Wise, Michael Ferguson, Derek King, Eric Diehr and David Dymesich
Workshop on Autonomous Mobile Service Robots, held at the
2016 International Joint Conference on Artificial Intelligence, NYC, July 2016
```

8.3.2 How do I create a good support ticket?

- Go to the Fetch Support Website and log in using your username and password
- Make sure the robot is not runstopped and then run the following commands on your local computer (Not the robot!)

```
sudo apt-get update && sudo apt-get install ros-indigo-fetch-tools
declare -x ROS_MASTER_URI="http://*RobotHostNameGoesHere*:11311"
fetch debug-snapshot
```

Note: You will need ros-indigo-fetch-tools version 0.1.2 or greater to use the debug-snapshot command

- Create a new support ticket and attach the zip file that was created
- Clearly state problem you are having in the ticket so we can better serve you

8.3.3 Why won't my robot do anything at all (charge, move, etc...)?

- Verify the red "breaker switch" on the back of the robot below the air vents is in the "ON" position

- If this did not solve your problem please contact Fetch Support

8.3.4 Why won't my robot turn on when I push the power button?

- If the charging light (#6 in figure below) is red Please try Charging the robot

- If this did not solve your problem please contact Fetch Support

8.3.5 Why won't my robot move when I use my PS3 joystick?

- If ring around the on button (#5 in figure below) is not illuminated please press the power button to turn on your robot (give it some time to boot and then try again)

- If there are no red lights illuminated on the front of the ps3 controller then press the round button in the center of the controller to turn on the controller (when it is ready to use it will vibrate) (Red lights will be in one of the 4 holes to the left of the charging port in the following picture.)

- Please verify that the runstop (seen below) has not been pressed.
  - If it is pressed twist it to turn it off

- If this did not solve your problem please contact Fetch Support

8.3.6 The robot will not move and/or slows down near obstacles, why?

- Please be aware, The robot's max speed is reduced when the laser sees an object directly in front of it. Despite this reduction the robot will always be able to travel at least 0.1 m/s even when the robot is almost touching an obstacle so it can not get locked down. This behavior is at the driver level.
- If you are autonomously navigating and the robot stops completely you should check your costmap and/or navigation algorithms
- If you are using the controller, it is on/connected and the robot stops please contact Fetch Support

8.3.7 I just sent a trajectory command to the arm and now it won't gravity compensate, help?

- Please verify that the trajectory command you sent to the arm was smoothed and doesn't exceed the velocity/acceleration limits of the arm. Otherwise when the arm exceeds its limits a breaker will trip cutting power to the arm
  - To reset the breakers please follow this guide (*Resetting Breakers*)
- If this did not solve your problem please contact Fetch Support

CHAPTER

NINE

LEGAL

9.1 Notices

9.1.1 FCC Notice

This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) This device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at their own expense.

Contains Transmitter Module FCC ID: PD97260H

9.1.2 Copyright Notices for Open Source Software

Fetch Robotics uses open-source software in Fetch and Freight. This software is licensed by the rights holders under the following licenses, among others: GNU General Public License (GPL2, GPL3), GNU Lesser Public License (LGPL), MIT License, zlib License, Apache2 license, and various forms and derivatives of the BSD license.

There are thousands of files installed by the Ubuntu installer, see http://www.ubuntu.com/about/about-ubuntu/licensing for details about licensing.

See the SICK TIM55x/56x/57x operating instructions for details of open source software usage in the SICK TIM571 scanning laser range finder.

9.1.3 Trademarks

Fetch, Freight, and Fetch Robotics are trademarks of Fetch Robotics Inc. Ubuntu is a trademark of Canonical Limited. All other trademarks and trade names are the property of their respective holders

9.2 License

9.3 Indices and tables

- genindex
- modindex
- search