Technical Manual

Maya Robot

Maya-ROS-UN

Santiago Araujo

Universidad Nacional de Colombia
Sede Bogotá
Departamento de ingeniería mecánica mecatrónica

Ernesto Córdoba Nieto
LabFabEx - Dima UN
May 2019

Maya - UN

Technical Manual

Contents

Prerequisites

Ubuntu

Because this project was written for Ros Kinetic, it needs Ubuntu Xenial (16.04). This documents assumes the user will be using this OS and so every command is written for it. Some familiarity with the command terminal is required.

Git

Every piece of code mentioned in this article is stored in a git repository. Students must become familiar with this CVS tool as it will be mentioned in several of the following sections. A gitlab user will be necessary to download the code and make contributions to it. *Gitlab* is the host currently containing LabFabEx's software.

Galiltools

Galil tools is a software that allows direct communication with the galil controller. The student that wants to contribute to this project should be familiarized with the GUI and functionality of this tool because it serves as a testing channel for commands, as well as a backup interface to the robot.

ROS Kinetic

Ros is the framework in which this project is written. It uses command and build tools of the framework, as well as several guidelines for it's file structure and coding style. All of the data transmitted in the system is done so via messages in topics and services, so the student should be familiarized with the core principles and functionalities of the framework.

Python

Python is the language in which most of the codebase is written. Every module except for ros-coms is written in Python 2.7.12 (Ros does not officially support Python 3).

NodeJS / Yarn / npm

NodeJS is used for the PRIA protocol. If communication with firebase is desired, node should be installed. NVM can be used to manage node versions. The ros-coms module has been tested with the latest LST, node v10.15.3 (npm v6.4.1). Although Yarn the preferred package manager in this project, you can also use npm. Commands for both will be written when necessary.

Typescript

The ros-coms module is written in typescript. Although contributing to ros-coms is outside of this document's scope, it may be helpful to understand how the maya_process package receives the process' information.

Installation

Galilsuite

Taken from galilmc:

1. Open a terminal
2. Get the Galil public key and import it into the package manager.

```
$ wget
http://www.galil.com/download/software/galilsuite/linux/galil_public_key.asc
$ gpg --no-default-keyring --keyring trustedkeys.gpg --import
galil_public_key.asc
```

3. Get the GalilSuite package and install it with the package manager.

```
$ wget
http://galil.com/download/software/galilsuite/linux/ubuntu_14.04/galilsu
ite_1.0.4_amd64.deb
$ sudo dpkg -i galilsuite_1.0.4_amd64.deb
```

4. GalilSuite can be launched from the terminal with the command "galilsuite" or from the system application launcher under GalilSuite.

```
$ galilsuite
```

Ros

Install ros Kinetic following the instructions on the official website.

rqt_rviz should also be installed

```
$ sudo apt install ros-kinetic-rqt-rviz
```

NodeJS

If you choose to install node through NVM (recommended), follow their installation instructions.

Else, run

```
$ sudo apt-get install curl python-software-properties
$ curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash -
$ sudo apt-get install nodejs
```

You will need to change the previous command if you want a different node version.

Yarn

To use Yarn commands, install it by running

```
$ npm install -g yarn
```

Typescript

```
$ npm install -g typescript
```

Maya-Un

The Maya-ROS-UN software is a Ros Stack. It's code is contained in several repositories hosted by *Gitlab*. Assuming your catkin workspace is on `~/catkin_ws`

```
$ cd ~/catkin_ws/src
$ git clone --recurse-submodules git@gitlab.com:LabFabEx/Maya-ROS-UN.git
```

Then, build your catkin workspace.

```
$ catkin build
```

Source your environment.

```
$ source ~/catkin_ws/devel/setup.bash
```

To build the *ros-coms* module

```
$ roscd maya_process/src/ros-coms
$ # if you're using yarn
```

```
$ yarn install
$ # else
$ npm install
$ tsc
```

gclib

To send commands to the physical robot, gclib must be installed. Installations instructions can be found on the official documentation.

Architecture Overview

The Maya-ROS-UN stack's architecture is based on the ros_control proposal. It attempts to decouple stages of functional logic of the robot similar to a Clean architecture pattern but applied to robots. Figure 1 shows a control pyramid, indicating dependencies downwards among layers i.e. maya_controller has no dependencies while maya_process depends on maya_controller that at the same time depends on maya_controller. This diagram hints which packages will run when a certain functionality is required.

Figure 2 shows in more detail the dependencies of the components of the stack. This shows how modules relate to each other and what modules will be needed to test certain functionalities, i. e. the firebase_bridge and process_controller can run independently from each other, but the primitive_server and robot_controller can't.

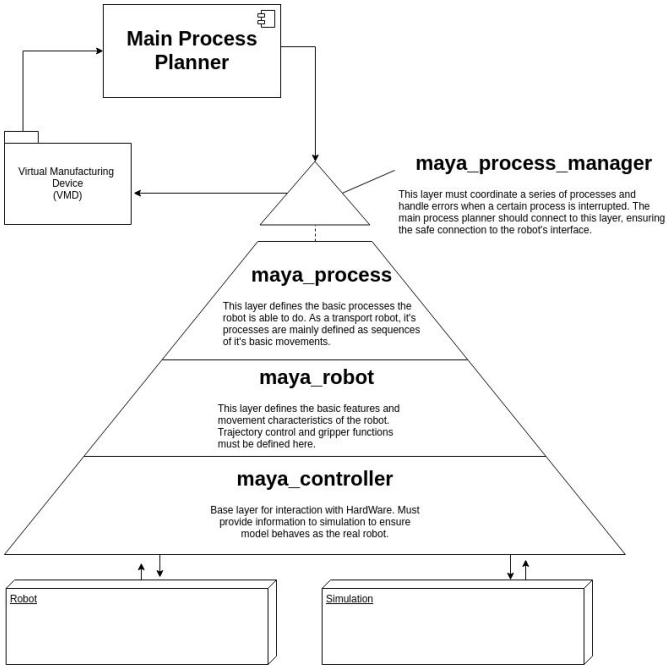

Figure 1: Maya control pyramid diagram

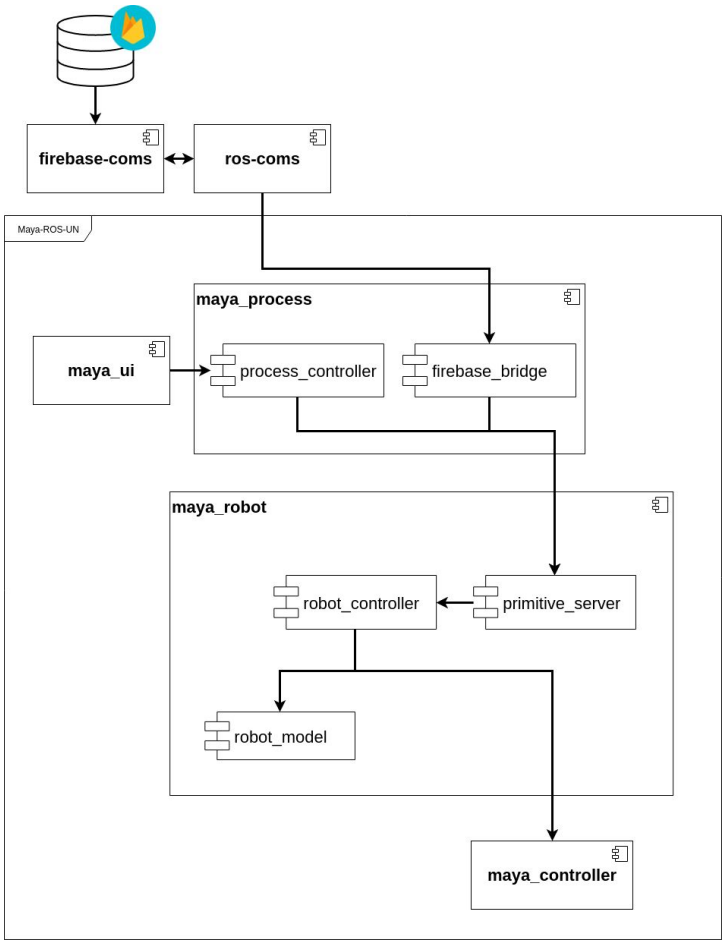

Figure 2: Module dependencies

Packaging

This section will show an overview of the packaging structure and files in this project. Binary files (.pyc), cache and some unimportant files are ignored.

Stack

Maya-ROS-UN is a ROS stack that contains all the code related to the Maya robot.

```
Maya-ROS-UN/
 ├─── maya_controller
 ├─── maya_process
 ├─── maya_robot
 ├─── maya_ui
 ├─── README.md
```

```
└──── stack.xml

4 directories, 2 files
```

Packages

maya_controller

This package contains the lowest level of control over the Maya robot. It's purpose is to execute articular commands, as well as read sensors and output events to modify the Maya virtual model. The file structure us show below.

```
Maya-ROS-UN/maya_controller/
├──── action
│     ├──── Home.action
│     └──── Move.action
├──── CMakeLists.txt
├──── launch
│     └──── controller.launch
├──── msg
│     └──── RawSensorData.msg
├──── package.xml
├──── README.md
├──── rviz
│     └──── urdf.rviz
├──── setup.py
├──── src
│     ├──── maya_controller
│     │     ├──── config
│     │     │     ├──── constants.py
│     │     │     ├──── __init__.py
│     │     │     └──── maya_events.py
│     │     ├──── hardware
│     │     │     ├──── _connection.py
│     │     │     ├──── _gripper.py
│     │     │     ├──── hardware_interface.py
│     │     │     ├──── home_helper.py
│     │     │     ├──── _motor.py
│     │     │     ├──── sensor_converter.py
│     │     │     └──── _sensor.py
│     │     ├──── __init__.py
│     │     └──── simulation
```

```
│   │       └── hw_simulator.py
│   └── virtual_maya.py
└── urdf
    ├── gripper.xacro
    └── maya_robot.xacro

10 directories, 24 files
```

Xacro is used to render the virtualized model of the robot. The virtual model can be upgraded through .stl files to represent more accurately the robot.

This package contains two folders to model the robot: One is the hardware folder and the other is simulation. Simulation provides a mock robot with basic direct kinematics to test the functionalities before running the code against the real robot. This helps accelerate the development process because students can develop and test their contributions without direct access to the machine. It also increases safety allowing tests in a safe virtual environment.

The `hw_simulator` takes robot commands and transforms them into joint_controller topics to simulate movement. The `hardware_interface` instead manages the action servers in charge of sending commands to the controller and verifying their result. Every file starting with an underscore in the hardware folder has access to a connection instance to the galil controller. This connection is disposed when the action is finished to avoid idling the connection to the controller.

The module `sensor_converter` has its own connection to the controller and constantly runs in its own thread. It periodically requests data from the sensors to update the virtual model.

The `config` folder has two files that concern the entire stack. The `constants` module contains operational information for the robot such as IP addresses and dynamic variables, and physical constants of the robot. Because the components on the robot may change due to replacement or breakage, this module contains a handy way to update the robot's physical structure's values. The `events` module contains values understood across the stack as state changes on the robot.

maya_robot

This package is responsible of handling the Maya model, implementing the logic for basic transport primitives. It also contains the code to calculate inverse kinematics and communicates them to the controller package to achieve movement.

```
Maya-ROS-UN/maya_robot/
├── action
```

```
│   └── Task.action
├── CMakeLists.txt
├── launch
│   └── robot.launch
├── msg
│   └── MayaPrimitive.msg
├── package.xml
├── README.md
├── setup.py
├── src
│   └── maya_robot
│       ├── __init__.py
│       ├── inv_kin_calculator.py
│       ├── model.py
│       ├── primitive_server.py
│       └── robot_controller.py
└── srv
    └── MayaCommand.srv

6 directories, 13 files
```

The package provides a service to perform queries of the state of the robot and send gripper commands. The primitive_server provides an action server that controls the *pick, place* and *approach* commands. It also receives and delegates a *home* command.

maya_process

This package holds two modules that regulate processes of the Maya robot. The `firebase_bridge` activate the topics that communicate with the `ros-coms` module. The `process_controller` provides a topic to send an array of primitives to the `primitive_server`, mainly used by the `maya_ui`.

```
Maya-ROS-UN/maya_process/
├── CMakeLists.txt
├── firebase_coms_launcher.sh
├── launch
│   └── process.launch
├── msg
│   └── ProcessQueue.msg
├── package.xml
├── setup.py
└── src
    ├── maya_process
```

```
     │    ├──── firebase_bridge.py
     │    ├──── __init__.py
     │    └──── process_controller.py
     └──── ros-coms

9 directories, 19 files
```

maya_ui

This package holds the rqt plugin for the Maya stack, as well as a *.perspective* file that configures rqt to show a collection of useful tools when using the robot.

```
Maya-ROS-UN/maya_ui/
├──── CMakeLists.txt
├──── package.xml
├──── plugin.xml
├──── resource
│    └──── MayaPlugin.ui
├──── rqt_config
│    └──── maya_tools.perspective
├──── scripts
│    └──── maya_ui.py
├──── setup.py
└──── src
     └──── maya_ui
          ├──── __init__.py
          ├──── maya_ui_module.py
          └──── ui_controller.py

5 directories, 10 files
```

ros-coms

The ros-coms module contains the code pertaining the connection to the RTDB that serves as a communication channel among robots and supervisors. It's cloned from its repository on the maya_process module to execute on that module's launch. It's details are outside this document's scope.

Configurations

The config.py module of the maya_controller package, provides a centralized module to store variables so that they can be easily modified by the end user.

maya_controller.config.constants.py

```
#!/usr/bin/env python
from math import pi

MAYA_ADDRESS = '192.168.1.156'
COMS_SERVER_ADDRESS = 'http://localhost:3000'


## ########################
##  WARNING
##  Modify carefully !!!
## ########################


#  Kinematic constants for the Robot

# PULSES_PER_M = 1_000_000.0 # python3 notation
PULSES_PER_M = 1000000.0 #1M
PULSES_PER_DEG = 371.06

MAYA_HEIGHT = 2.35
MAYA_LENGTH =  2.3

A_0 = 0
A_MAX = 2*pi/3

B_0 = 0.575
C_0 = 0.565
B_MAX = 2.260
C_MAX = 2.300

DELTA = 0.30 # distance from a axis to b axis
SIGMA = 0.20 # gripper height


#  Dynamic constants for the Robot
```

```
ACCEL_A = 1024
ACCEL_B = 5120
ACCEL_C = 5120

DCEL_A = 1024
DCEL_B = 5120
DCEL_C = 5120

SP_A = 1000
SP_B = 45000
SP_C = 45000

########################
```

Running the Maya Software

The Maya stack has several launch options to allow modular testing of each package.

roslaunch

Every package, except maya_ui, contains a .launch file that receives parameters to exclude or include other launch scripts. The `roslaunch` command is run through a terminal. The command structure is as follows

```
$ roslaunch {package} {launch_file} --{argument}:={value}
```

maya_controller

```
$ roslaunch maya_controller controller.launch
```

Argument	Default	Description
gui	false	If true, opens rviz gui tool to modify joint_states
core_debug	false	If true, skips rviz launch.
rvizconfig	$(find maya_controller) /rviz/urdf.rviz	File path to the rviz config

sim_mode	true	If true, launches simulated robot nodes only. Else, launches `hw_interface` and `sensor_converter` modules
skip_home	false	If true, disables the initial requirement of the robot to be homed. **Only use if a home operation has already been executed after the last reset of the controller.**

maya_robot

Launches the maya_robot nodes and calls the maya_controller launch, passing along the specified parameters.

```
$ roslaunch maya_robot robot.launch
```

Argument	Default	Description
core_debug	false	If true, skips rviz launch.
sim_mode	true	If true, launches simulated robot nodes only. Else, launches `hw_interface` and `sensor_converter` modules
skip_home	false	If true, disables the initial requirement of the robot to be homed. **Only use if a home operation has already been executed after the last reset of the controller.**

maya_process

Launches the maya_process nodes and calls the maya_robot launch, passing along the specified parameters.

```
$ roslaunch maya_process process.launch
```

Argument	Default	Description
core_debug	true	If true, skips rviz launch.
sim_mode	true	If true, launches simulated robot nodes only. Else, launches `hw_interface` and `sensor_converter` modules
skip_home	false	If true, disables the initial requirement of the robot to be homed. **Only use if a home operation has already been executed after the last reset of the controller.**
custom_gui	true	If true, launches rqt, loaded with the `maya_ui` perspective.
skip_firebase	false	If false, launches the ros-coms module along the nodes of the package

maya_ui

The Maya plugin can be launched independently from the rest of the nodes. To run as a standalone plugin run the command

```
$ rqt --standalone maya_ui
```

Yarn / npm

The ros-comms module is launched with a package manager. To run independently, set skip_firebase to true and run

```
$ roscd maya_process/src/ros-coms
$ yarn start
$ # or
$ npm run start
```

Arguments can be passed to the previous commands in the format

```
$ ARG=VALUE yarn start
$ # or
```

```
$ ARG=VALUE npm run start
```

Argument	Default	Description
PORT	3000	The port where the server will run. If modified for Maya, the config `COMS_SERVER_ADDRESS` should be modified as well.
IP	ws://localhost:9090	The websocket address for the rosbridge server
NAME	MAYA	The identifier name the robot will use in *FireStore*

Galil Commands

Commands are sent to the Gail controller via *gclib.* A complete overview of the supported commands can be found on the DMC-41x3 controller reference manual, or the document *DMC-41x3 Firmware Command Reference.* The commands used are summarized in the table below.

Command	Purpose
CB1 / **SB1**	Activate / Deactivate gripper
PA	Send an absolute position to the motors
PR	Send an relative position to the motors
JG	Move with specified speeds
BG	Begin movement
SH	Turns on servos
ST	Stop movement
MO	Motors off
AB	Aborts movement

SP / **AC** / **DC**	Sets dynamic operation values for motors: Maximum speed, acceleration and deceleration.
lim={limit_sensor};lim=	Reads {limit_sensor} and returns 1 if is active, 0 otherwise. {limit_sensor} follows the pattern _L{F or R}{motor} i.e. _LFA, _LRA
TV	Gets velocity of motors

Direct Kinematics

Direct kinematics are calculated simplifying the robot as shown in figure 7. The terms *α*, *b* and *c*, refer to the articular variables of the robot controlled by motors *a*, *b* and *c* respectively. The height of the robot and the arm's length are constants, denoted by the capital *H* and *D,* and the distances Δ and Φ as shown in figure 8.

In direct kinematics, the articular values are known and we calculate the *x*, *y*, *z* coordinates of the end effector. The orientation of the gripper is the same orientation as the arm, rotated *α* from the original frame of reference. It can't rotate along the other axes. Because the gripper doesn't depend on its orientation to operate, it's mostly ignored.

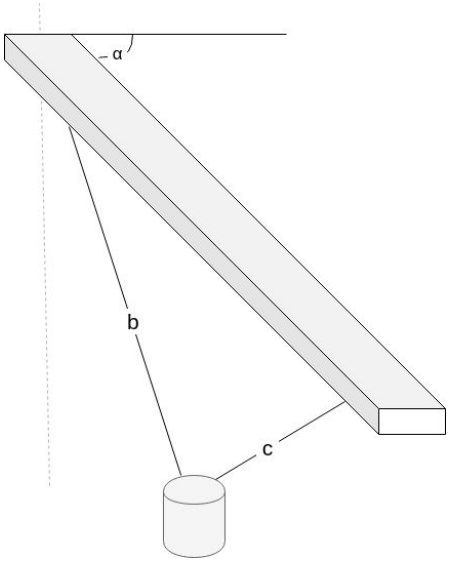

Figure 7: Maya diagram

Knowing the values of *b*, *c* and *D* we can calculate angle *β* using the cosine law formula.

$$\beta = cos^{-1}\left(\frac{(D-\Delta)^2+b^2-c^2}{2(D-\Delta)\cdot b}\right)$$

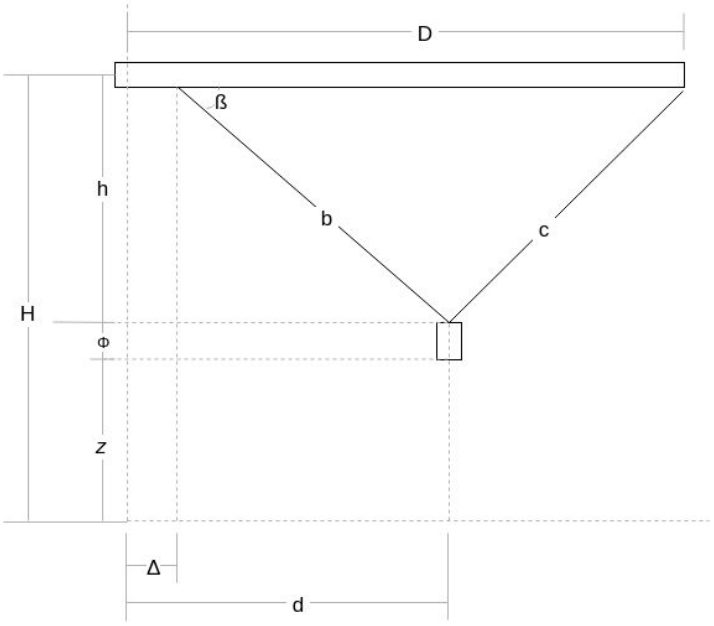

Figure 8: Maya side view

Once we have *β,* we can find *d* and *h* using trigonometry.

$$d = b \cdot cos(\beta) + \Delta$$
$$z = H - b \cdot sin(\beta) - \Phi$$

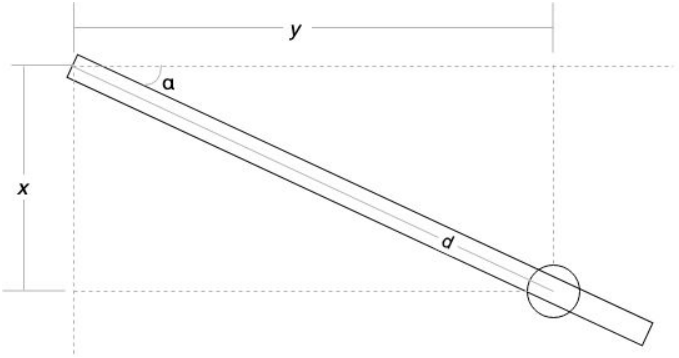

Figure 9: Maya top view

Having found the distance from the gripper to the base, we can calculate the missing coordinates.

$$x = d \cdot sin(\alpha)$$
$$y = d \cdot cos(\alpha)$$

So we have as a result:

$$x = \left(b \cdot \frac{D^2+b^2-c^2}{2Db} + \Delta\right) \cdot sin(\alpha)$$
$$y = \left(b \cdot \frac{D^2+b^2-c^2}{2Db} + \Delta\right) \cdot cos(\alpha)$$
$$z = H - b \cdot sin\left(cos^{-1}\left(\frac{D^2+b^2-c^2}{2Db}\right)\right) - \Phi$$

Inverse Kinematics

In inverse kinematics, we calculate the the articular values for a goal given set of coordinates *x*, *y*, *z*.

From figure 9 we can deduce

$$\alpha = arctan(x, y)$$
$$d = \sqrt{x^2+y^2}$$

From there, in figure 8 we can see the following relations

$$b = \sqrt{(H - z + \Phi)^2 + (d - \Delta)^2}$$

$$c = \sqrt{(H - z + \Phi)^2 + (D - d)^2}$$

Workspace Validations

Although the Maya's robot workspace is not properly calculated, some validations take place before sending the goal to he robot. The following code prevents unreachable goals from being sent to the robot before all the calculations are made.

```
def validate_ws(alpha, d, z):

    if z > c.MAYA_HEIGHT:
        raise ValueError('requested gripper position too high (z=%s)' % z)
    if z < 0:
        raise ValueError('requested gripper position too low (z=%s)' % z)
    if d > c.MAYA_LENGTH-.3:
        raise ValueError('requested gripper position too far (d=%s)' % (d+.3))
    if d < 0:
        raise ValueError(
```

```
            'requested gripper position too close to base (d=%s)' % (d+.3))
```

Once the calculations are complete, an additional validation is performed at the controller level.

```
def validate_ws(a_dest, l1_dest, l2_dest):

    if a_dest < c.A_0 or a_dest > m.degrees(c.A_MAX):
        raise ValueError('Can\'t reach a :%f' % a_dest)
    if (l1_dest > c.B_MAX) or (l1_dest < c.B_0):
        raise ValueError('Can\'t reach b :%f' % l1_dest)
    if (l2_dest > c.C_MAX) or (l2_dest < c.C_0):
        raise ValueError('Can\'t reach c :%f' % l2_dest)
```

This allows for a safer operation of the robot.

Communications

This project follows the Ros' [communications pattern guidelines](#).

Topics

`/maya_event`

Used to register state changes in the robot.

```
Type: std_msgs/String

Publishers:
 * /hw_simulator
 * /home
 * /hardware_interface

Subscribers:
 * /robot_controller
 * /maya_dashboard
```

/maya_move

Used as a topic interface to the primitive server.

```
Type: maya_robot/MayaPrimitive

Publishers: None

Subscribers:
 * /robot_controller
```

/maya_position

Updates the current position of the end-effector on the virtual maya model.

```
Type: geometry_msgs/Point

Publishers:
 * /hw_simulator
 * /sensor_converter

Subscribers:
 * /robot_controller
 * /maya_dashboard
```

/hardware_sensor

Used by the `sensor_converter` module to emit sensor read results.

```
Type: maya_controller/RawSensorData

Publishers:
 * /sensor_converter

Subscribers:
 * /sensor_converter
```

/send_task

Topic used by the `ui_controller` to send a task to the `process_controller`

```
Type: maya_robot/MayaPrimitive

Publishers:
 * /maya_dashboard

Subscribers:
 * /process_controller
```

`/start_queue`

Topic used by the `ui_controller` to send a list of tasks to the `process_controller`

```
Type: maya_process/ProcessQueue

Publishers:
 * /maya_dashboard

Subscribers:
 * /process_controller
```

`/stop_queue`

Used by the `ui_controller` to cancel any pending movements of the robot, including a HomeAction

```
Type: std_msgs/Empty

Publishers:
 * /maya_dashboard

Subscribers:
 * /process_controller
```

`/step_command`

Used to communicate a stringified json carrying a task's data.

```
Type: std_msgs/String

Publishers: ros-coms

Subscribers:
 * /firebase_bridge
```

`/step_result`

Used to communicate a stringified json carrying a task's result.

```
Type: std_msgs/String

Publishers:
 * /firebase_bridge

Subscribers: ros-coms
```

Services

`/gripper`

Used by the `robot_controller` to control the gripper state. When called, an event is published with the new gripper state to `/maya_events`. Takes a boolean that represents the state of the gripper, open if true, closed otherwise.

`/maya_command`

Service that exposes all the primitives for the robot Maya. It also queries if the robot can be teleoperated (is homed and not moving) and gives access to the gripper methods. It receives a MayaPrimitive in a task parameter, where the supported commands are can_teleop, home, gripper, pick, place and approach.

Actions

Actions provide a handle on long running requests. The processes that the Maya robot needs time to perform are homing and moving.

/`home`

This action server receives a HomeAction and returns a boolean indicating if the homing process has completed successfully.

/`move`

The move action server receives a MoveAction which contains the articular values needed.

/`task`

This server uses both previous servers to create a higher level abstraction for the maya allowing for pick and place operations. It receives a TaskAction which contains a MayaPrimitive to execute.

Troubleshooting

- `catkin build` not working

Try using `catkin_make`. Generally it's not a good idea to mix both commands, and `catkin build` is the preferred method. However experience shows some workspaces may need the `make` command.

- Cannot connect to the robot

Check if connection can be established through *galisuite*. Try restarting the controller if necessary. Sometimes the controller fails to get an IP assigned. After a reset this is usually solved.

Contributing

Style guide

This project tries to follow the official python styleguide. Use this guide for naming classes, methods, packages, etc. Keep your code pythonic using Python3 compliant features.

Git workflow

It's recommended for students to familiarize themselves with a git workflow such as git flow to achieve seamless integration among developers. Atlassian has a good guide for it.

master and develop branches are protected, so to scale a version up, a pull request should be submitted from the feature branch.