These notes and instructions are from the Galen Robot Admittance Controller team design meeting on Thursday, March 16th, 2023. By Brevin Banks bbanks15@jhu.edu

Because the Zoom meeting recording failed to capture the video and audio from the meeting, the important topics shared by Adnan Munawar have been captured and summarized below in writing.

The following code and example output images were produced on a virtual machine Linux platform running Ubuntu 20.04 with ROS Noetic installed and the user is name 'robotics'. AMBF has been installed and the Galen Robot ADF files have been built, made, and located on the home drive in a folder called Galen_AMBF_Simulation outside of the /ambf folder.

# Contents

a. AMBF General Installation

Regarding the installation and instructions on AMBF in general see the GitHub page as well as the installation and use wiki

https://github.com/WPI-AIM/ambf

https://github.com/WPI-AIM/ambf/wiki

If you have access to the repo at

https://github.com/jtzhangLogan/Galen_AMBF_Simulation

you can use the Galen Robot in AMBF and make use of controlling it or updating the dynamics with the instructions here.

# 1. Running the Galen Robot in AMBF:

Ensure ROS is running in the background in a separate terminal. Run:

Roscore

```
robotics@ubuntu:~$ roscore
... logging to /home/robotics/.ros/log/c4595706-c458-11ed-8fa6-3dd695f6de02/roslaunch-ubuntu-71659.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:46823/
ros_comm version 1.15.15


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.15

NODES

auto-starting new master
process[master]: started with pid [71674]
ROS_MASTER_URI=http://ubuntu:11311/

setting /run_id to c4595706-c458-11ed-8fa6-3dd695f6de02
process[rosout-1]: started with pid [71691]
started core service [/rosout]
```

Open another terminal for running AMBF. If needed Run:

source ambf/build/devel/setup.bash

To run the Galen robot, ensure you are in the folder (corresponding to your OS):

cd ambf/bin/lin-x86_64

To run the Galen robot with the skull and sensors, assuming your file path has Galen_AMBF_Simulation at the home directory, run:

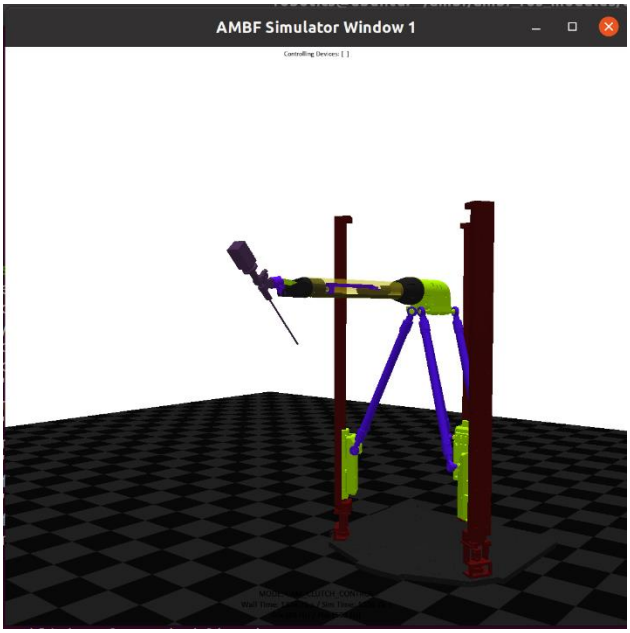
./ambf_simulator --launch_file ~/Galen_AMBF_Simulation/ADF/launch.yaml -l 0,1,2

To run the Galen robot with just the robot run:

./ambf_simulator --launch_file ~/Galen_AMBF_Simulation/ADF/launch.yaml -l 0

```
robotics@ubuntu:~/ambf/bin/lin-x86_64$ ./ambf_simulator --launch_file ~/Galen_AMBF_Simulation/ADF/launch.yaml -l 0

-----------------------------------------------------------

ASYNCHRONOUS MULTI-BODY FRAMEWORK SIMULATOR (AMBF Simulator)


               (http://practicepoint.wpi.edu)
                 (Copyright 2019-2021)
-----------------------------------------------------------

STARTUP COMMAND LINE OPTIONS:

ambf_simulator Command Line Options:
  -h [ --help ]                         Show help
  -n [ --ndevs ] arg (=0)               Number of Haptic Devices to Load
  -i [ --load_devices ] arg             Index number of devices to load which is
                                        specified in input_device.yaml
  -f [ --enable_forces ] arg (=0)       Enable Force Feedback on Haptic Devices
  -p [ --phx_frequency ] arg (=1000)    Physics Update Frequency (default: 1000
                                        Hz)
  -d [ --htx_frequency ] arg (=1000)    Haptics Update Frequency (default: 1000
                                        Hz)
  -t [ --fixed_phx_timestep ] arg (=0) Use Fixed Time-Step for Physics
                                        (default: False)
  -f [ --fixed_htx_timestep ] arg (=0) Use Fixed Time-Step for Haptics
                                        (default: False)
  --override_max_comm_freq arg          Override the maximum publishing
                                        frequency for all afObjects (default:
                                        1000 Hz)
  --override_min_comm_freq arg          Override the minimum publishing
                                        frequency for all afObjects (default: 50
                                        Hz)
  -g [ --show_gui ] arg (=1)            Show GUI
  --ns arg                              Global namespace prefix for ROS
                                        Communication
  -s [ --sim_speed_factor ] arg (=1)    Override the speed of "NON REAL-TIME"
                                        simulation by a specified factor
                                        (Default 1.0)
  --plugins arg                         Simulator plugins to load, .e.g.
                                        --plugins <plugin1_filepath>,
                                        <plugin2_filepath> loads plugin1 and
                                        plugin2 simualtor plugin
  --launch_file arg                     Launch file path to load (default:
                                        <ROOT_PATH>/ambf_models/descriptions/lau
                                        nch.yaml
```

```
  -a [ --load_multibody_files ] arg    Description Filenames of Multi-Body(ies)
                                       to Launch, .e.g. -a <path>/test.yaml,
                                       <another_path>/test2.yaml will load
                                       multibodies test.yaml and test2.yaml if
                                       they are valid files
  -l [ --load_multibodies ] arg        Index of Multi-Body(ies) to Launch,
                                       .e.g. -l 1,2,3 will load multibodies at
                                       indexes 1,2,3. See launch.yaml file


-------------------------------------------------------------



GLFW VERSION: 3.2.1 X11 GLX EGL clock_gettime /dev/js
WARNING! For File "/home/robotics/Galen_AMBF_Simulation/ADF/launch.yaml", ADF version not defined thus assuming VERSION_1_0
WARNING! For File "/home/robotics/Galen_AMBF_Simulation/ADF/./world/world.yaml", ADF version not defined thus assuming VERSION_1_0
WARNING! For File "/home/robotics/Galen_AMBF_Simulation/ADF/./input_devices/input_devices.yaml", ADF version not defined thus assuming VERSION_1_0
INFO! INITIALIZING ROS NODE HANDLE
INFO! Thread Joined: Plane
INFO! Thread Joined: light_left
INFO! Thread Joined: default_camera
INFO! Thread Joined: World
WARNING! For File "/home/robotics/Galen_AMBF_Simulation/ADF/./galen.yaml", ADF version not defined thus assuming VERSION_1_0
INFO! Thread Joined: Base
INFO! Thread Joined: Carriage1
INFO! Thread Joined: Carriage2
INFO! Thread Joined: Carriage3
INFO! Thread Joined: Chassis
INFO! Thread Joined: Endoscope35degreeinREMS
INFO! Thread Joined: EndoscopeTip
INFO! Thread Joined: MobilePlatform
INFO! Thread Joined: RollArmBase
INFO! Thread Joined: TiltDistalLinkageandForceSensor
WARNING! COMMUNICATION TYPE FOR OBJECT NAMED roll_joint OF TYPE: JOINT NOT IMPLEMENTED YET. IGNORING.
WARNING! COMMUNICATION TYPE FOR OBJECT NAMED carriage1_joint OF TYPE: JOINT NOT IMPLEMENTED YET. IGNORING.
WARNING! COMMUNICATION TYPE FOR OBJECT NAMED carriage2_joint OF TYPE: JOINT NOT IMPLEMENTED YET. IGNORING.
WARNING! COMMUNICATION TYPE FOR OBJECT NAMED carriage3_joint OF TYPE: JOINT NOT IMPLEMENTED YET. IGNORING.
WARNING! COMMUNICATION TYPE FOR OBJECT NAMED TiltDistalLinkageandForceSensor-Endoscope35degreeinREMS OF TYPE: JOINT NOT IMPLEMENTED YET. IGNORING.
WARNING! COMMUNICATION TYPE FOR OBJECT NAMED tilt_joint OF TYPE: JOINT NOT IMPLEMENTED YET. IGNORING.
Failed to load plugin libGalenControlPlugin.so: libGalenControlPlugin.so: cannot open shared object file: No such file or directory
WatchDog expired, Resetting "Base" command
gravity ON:
gravity ON:
gravity OFF:
INFO! TOTAL ACTIVE COMM INSTANCES: 14
INFO! WAITING FOR ALL COMM INSTANCES TO UNREGISTER ...
       INFO! REMAINING ACTIVE COMMs: 13
       INFO! REMAINING ACTIVE COMMs: 4
```




The robot will appear in a window similar to this where visualization of control responses will be available.

# 2. AMBF RigidBodyCmd

Observe how AMBF publishes ROS messages for a robot with RigidBodyCmd. In a separate terminal (with the AMBF simulation closed) run:

rosmsg show ambf_msgs/RigidBodyCmd

```
robotics@ubuntu:~$ rosmsg show ambf_msgs/RigidBodyCmd
int8 TYPE_FORCE=0
int8 TYPE_POSITION=1
int8 TYPE_VELOCITY=2
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
int8 cartesian_cmd_type
geometry_msgs/Pose pose
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
geometry_msgs/Wrench wrench
  geometry_msgs/Vector3 force
    float64 x
    float64 y
    float64 z
  geometry_msgs/Vector3 torque
    float64 x
    float64 y
    float64 z
geometry_msgs/Twist twist
  geometry_msgs/Vector3 linear
    float64 x
    float64 y
    float64 z
  geometry_msgs/Vector3 angular
    float64 x
    float64 y
    float64 z
int8[] joint_cmds_types
float32[] joint_cmds
bool publish_children_names
bool publish_joint_names
bool publish_joint_positions
```

This will display the different types of messages publishable to ROS from AMBF. These can be used to program in python, MATLAB, etc. messages that control the robot in AMBF. Notice that there are 3 types at the top. TYPE_FORCE, TYPE_POSITION, AND TYPE_VELOCITY. These indicate what type of control is being applied to a specific joint of the robot when commands are applied to the cartesian_cmd_type. (0 for force control, 1 for position control, and 2 for velocity control).

This is a test example for using the RigidBodyCmd message publisher. This will display the different controllable perspectives of the robot. It should show the cartesian_cmd_type which represents the type of control being applied to the robot. run:

rostopic pub /test ambf_msgs/RigidBodyCmd "header:

```
robotics@ubuntu:~$ rostopic pub /test ambf_msgs/RigidBodyCmd "header:
  seq: 0
  stamp: {secs: 0, nsecs: 0}
  frame_id: ''
cartesian_cmd_type: 1
pose:
  position: {x: 0.0, y: 0.0, z: 0.0}
  orientation: {x: 0.0, y: 0.0, z: 0.0, w: 0.0}
wrench:
  force: {x: 0.0, y: 0.0, z: 0.0}
  torque: {x: 0.0, y: 0.0, z: 0.0}
twist:
  linear: {x: 0.0, y: 0.0, z: 0.0}
  angular: {x: 0.0, y: 0.0, z: 0.0}
joint_cmds_types: [0,0,0,0]
joint_cmds: [0,0,0,0^C
publish_children_names: false
publish_joint_names: false
publish_joint_positions: false"
```

## 3. Python Controller

There is a convenient python client for controlling the robot joints that publishes using the messages from RigidBodyCmd. You can use this controller and echo the published messages to observe how the controller implemented the command. These messages can be used in the ROS toolbox in MATLAB to publish to the robot.

Start by navigating to the python controller path:

cd ambf/ambf_ros_modules/examples/object_control_example/

We will run the python controller in this location. Using -h you can see the relevant and possible commands:

python3 -h

```
robotics@ubuntu:~$ cd ambf/ambf_ros_modules/examples/object_control_example/
robotics@ubuntu:~/ambf/ambf_ros_modules/examples/object_control_example$ python3 -h
usage: python3 [option] ... [-c cmd | -m mod | file | -] [arg] ...
Options and arguments (and corresponding environment variables):
-b     : issue warnings about str(bytes_instance), str(bytearray_instance)
         and comparing bytes/bytearray with str. (-bb: issue errors)
-B     : don't write .pyc files on import; also PYTHONDONTWRITEBYTECODE=x
-c cmd : program passed in as string (terminates option list)
-d     : debug output from parser; also PYTHONDEBUG=x
-E     : ignore PYTHON* environment variables (such as PYTHONPATH)
-h     : print this help message and exit (also --help)
-i     : inspect interactively after running script; forces a prompt even
         if stdin does not appear to be a terminal; also PYTHONINSPECT=x
-I     : isolate Python from the user's environment (implies -E and -s)
-m mod : run library module as a script (terminates option list)
-O     : remove assert and __debug__-dependent statements; add .opt-1 before
         .pyc extension; also PYTHONOPTIMIZE=x
-OO    : do -O changes and also discard docstrings; add .opt-2 before
         .pyc extension
-q     : don't print version and copyright messages on interactive startup
-s     : don't add user site directory to sys.path; also PYTHONNOUSERSITE
-S     : don't imply 'import site' on initialization
-u     : force the stdout and stderr streams to be unbuffered;
         this option has no effect on stdin; also PYTHONUNBUFFERED=x
-v     : verbose (trace import statements); also PYTHONVERBOSE=x
         can be supplied multiple times to increase verbosity
-V     : print the Python version number and exit (also --version)
         when given twice, print more information about the build
-W arg : warning control; arg is action:message:category:module:lineno
         also PYTHONWARNINGS=arg
-x     : skip first line of source, allowing use of non-Unix forms of #!cmd
-X opt : set implementation-specific option. The following options are available:

         -X faulthandler: enable faulthandler
         -X showrefcount: output the total reference count and number of used
             memory blocks when the program finishes or after each statement in the
             interactive interpreter. This only works on debug builds
         -X tracemalloc: start tracing Python memory allocations using the
             tracemalloc module. By default, only the most recent frame is stored in a
             traceback of a trace. Use -X tracemalloc=NFRAME to start tracing with a
             traceback limit of NFRAME frames
         -X showalloccount: output the total count of allocated objects for each
             type when the program finishes. This only works when Python was built with
             COUNT_ALLOCS defined
         -X importtime: show how long each import takes. It shows module name,
             cumulative time (including nested imports) and self time (excluding
             nested imports). Note that its output may be broken in multi-threaded
             application. Typical usage is python3 -X importtime -c 'import asyncio'
         -X dev: enable CPython's "development mode", introducing additional runtime
             checks which are too expensive to be enabled by default. Effect of the
             developer mode:
               * Add default warning filter, as -W default
               * Install debug hooks on memory allocators: see the PyMem_SetupDebugHooks() C function
               * Enable the faulthandler module to dump the Python traceback on a crash
               * Enable asyncio debug mode
               * Set the dev_mode attribute of sys.flags to True
               * io.IOBase destructor logs close() exceptions
         -X utf8: enable UTF-8 mode for operating system interfaces, overriding the default
             locale-aware mode. -X utf8=0 explicitly disables UTF-8 mode (even when it would
             otherwise activate automatically)
         -X pycache_prefix=PATH: enable writing .pyc files to a parallel tree rooted at the
             given directory instead of to the code tree

--check-hash-based-pycs always|default|never:
    control how Python invalidates hash-based .pyc files
file   : program read from script file
-      : program read from stdin (default; interactive mode if a tty)
arg ...: arguments passed to program in sys.argv[1:]

Other environment variables:
PYTHONSTARTUP: file executed on interactive startup (no default)
PYTHONPATH   : ':'-separated list of directories prefixed to the
               default module search path.  The result is sys.path.
PYTHONHOME   : alternate <prefix> directory (or <prefix>:<exec_prefix>).
               The default module search path uses <prefix>/lib/pythonX.X.
PYTHONCASEOK : ignore case in 'import' statements (Windows).
PYTHONUTF8: if set to 1, enable the UTF-8 mode.
PYTHONIOENCODING: Encoding[:errors] used for stdin/stdout/stderr.
PYTHONFAULTHANDLER: dump the Python traceback on fatal errors.
PYTHONHASHSEED: if this variable is set to 'random', a random value is used
   to seed the hashes of str and bytes objects.  It can also be set to an
   integer in the range [0,4294967295] to get hash values with a
   predictable seed.
PYTHONMALLOC: set the Python memory allocators and/or install debug hooks
   on Python memory allocators. Use PYTHONMALLOC=debug to install debug
   hooks.
PYTHONCOERCECLOCALE: if this variable is set to 0, it disables the locale
   coercion behavior. Use PYTHONCOERCECLOCALE=warn to request display of
   locale coercion and locale compatibility warnings on stderr.
PYTHONBREAKPOINT: if this variable is set to 0, it disables the default
   debugger. It can be set to the callable of your debugger of choice.
PYTHONDEVMODE: enable the development mode.
PYTHONPYCACHEPREFIX: root directory for bytecode cache (pyc) files.
```

We will create an instance of the python script control_object.py in a second. Using -h you can see the relevant and possible commands associated with this controller script:

python3 control_object.py -h

```
robotics@ubuntu:~/ambf/ambf_ros_modules/examples/object_control_example$ python3 control_object.py -h
usage: control_object.py [-h] [-o OBJ_NAME] [-c ENABLE_CARTESIAN_CONTROL]
                         [-j ENABLE_JOINT_CONTROL] [-a CLIENT_NAME]
                         [--ixyz INITIAL_XYZ] [--irpy INITIAL_RPY]
                         [--rxyz RANGE_XYZ] [--rrpy RANGE_RPY]
                         [--res RESOLUTION]

optional arguments:
  -h, --help            show this help message and exit
  -o OBJ_NAME           Specify AMBF Obj Name
  -c ENABLE_CARTESIAN_CONTROL
                        Enable Control of Cartesian Space
  -j ENABLE_JOINT_CONTROL
                        Enable Control of Joint Space
  -a CLIENT_NAME        Client Name
  --ixyz INITIAL_XYZ    Initial XYZ
  --irpy INITIAL_RPY    Initial RPY
  --rxyz RANGE_XYZ      Range XYZ
  --rrpy RANGE_RPY      Range RPY
  --res RESOLUTION      Resolution
```

You can see -o will specify the object (the most parent robot link you'd wish to control and following children. Most likely the base of the robot). You can choose which link to use. The Base of the Galen robot is chosen in this case. 'Base'. -j will enable joint space control with 1 setting the node to true. -c will enable cartesian space control with 0 setting the node to false. In our controller design we are intending to implement a joint space controller with velocity control. To initiate the python controller instance run:

python3 control_object.py -o Base -j 1 -c 0

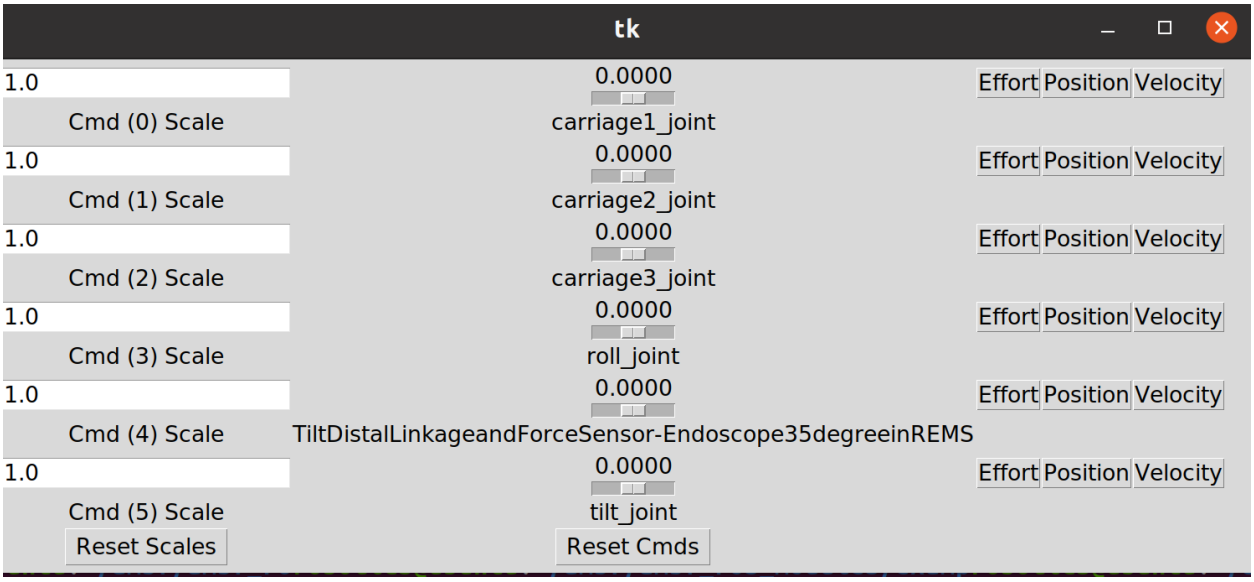
```
robotics@ubuntu:~/ambf/ambf_ros_modules/examples/object_control_example$ python3 control_object.py -o Base -j 1 -c 0
Specified Arguments
Namespace(client_name=None, enable_cartesian_control='0', enable_joint_control='1', initial_rpy='0, 0, 0', initial_xyz='0, 0
, 0', obj_name='Base', range_rpy='3.14', range_xyz='1', resolution='0.001')
Found Common Object Namespace as:  /ambf/env/
WARNING FOUND  2 WITH MATCHING NAME:
/ambf/env/Base
/ambf/env/RollArmBase
```

The output may be confused by the term 'Base' as many robots often have a link of this name in the ambf/env/ directory. To be more explicit you can use /ambf/env/Base to specify the desired body. Run:

python3 control_object.py -o /ambf/env/Base -j 1 -c 0

```
robotics@ubuntu:~/ambf/ambf_ros_modules/examples/object_control_example$ python3 control_object.py -o /ambf/env/Base -j 1 -c 0
Specified Arguments
Namespace(client_name=None, enable_cartesian_control='0', enable_joint_control='1', initial_rpy='0, 0, 0', initial_xyz='0, 0, 0', obj
_name='/ambf/env/Base', range_rpy='3.14', range_xyz='1', resolution='0.001')
Found Common Object Namespace as:  /ambf/env/
```

A tk window will open up with buttons that can change the type of control as well as the angle and scale of command. By default, effort is selected as the control scheme.




To see the different topics that are affected by these controls you can look at the rostopic lists. Run:

rostopic list

```
robotics@ubuntu:~/ambf/ambf_ros_modules/examples/object_control_example$ rostopic list
/ambf/env/Base/Command
/ambf/env/Base/State
/ambf/env/Carriage1/Command
/ambf/env/Carriage1/State
/ambf/env/Carriage2/Command
/ambf/env/Carriage2/State
/ambf/env/Carriage3/Command
/ambf/env/Carriage3/State
/ambf/env/Chassis/Command
/ambf/env/Chassis/State
/ambf/env/Endoscope35degreeinREMS/Command
/ambf/env/Endoscope35degreeinREMS/State
/ambf/env/EndoscopeTip/Command
/ambf/env/EndoscopeTip/State
/ambf/env/MobilePlatform/Command
/ambf/env/MobilePlatform/State
/ambf/env/Plane/Command
/ambf/env/Plane/State
/ambf/env/RollArmBase/Command
/ambf/env/RollArmBase/State
/ambf/env/TiltDistalLinkageandForceSensor/Command
/ambf/env/TiltDistalLinkageandForceSensor/State
/ambf/env/World/Command
/ambf/env/World/Command/Reset
/ambf/env/World/Command/Reset/Bodies
/ambf/env/World/State
/ambf/env/World/point_cloud
/ambf/env/World/point_cloud/radius
/ambf/env/cameras/default_camera/Command
/ambf/env/cameras/default_camera/State
/ambf/env/lights/light_left/Command
/ambf/env/lights/light_left/State
/rosout
/rosout_agg
```

The topic /ambf/env/Base/Command in this example is the parent topic that sends subroutines to the Carriages and other joints. If you try to observe the Commands for children you may not be able to see anything, but observing the Command for the base will show the control type for each child joint and the current input. It should be noted that in order to have the desired inputs actually move the joint children inside the parent 'Base' it is required that you publish to /ambf/env/Base/Command. Observe the Command of the Base. Run:

```
rostopic echo /ambf/env/Base/Command
```

```
robotics@ubuntu:~/ambf/ambf_ros_modules/examples/object_control_example$ rostopic echo /ambf/env/Base/Command
header:
  seq: 201453
  stamp:
    secs: 1679017024
    nsecs: 755486488
  frame_id: ''
cartesian_cmd_type: 0
pose:
  position:
    x: 0.0
    y: 0.0
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.0
    w: 0.0
wrench:
  force:
    x: 0.0
    y: 0.0
    z: 0.0
  torque:
    x: 0.0
    y: 0.0
    z: 0.0
twist:
  linear:
    x: 0.0
    y: 0.0
    z: 0.0
  angular:
    x: 0.0
    y: 0.0
    z: 0.0
joint_cmds_types: [2, 2, 2, 2, 2, 2]
joint_cmds: [0.14710000157356262, 0.0, -0.29409998655319214, -0.3528999984264374, -0.29409998655319214, 0.0]
publish_children_names: True
publish_joint_names: True
publish_joint_positions: True
```

You can observe the state of the Base and other joints by looking at their State topics.

For the Base run:

```
rostopic echo /ambf/env/Base/State
```

```
^Crobotics@ubuntu:~/ambf/ambf_ros_modules/examples/object_control_example$ rostopic echo /ambf/env/Base/State
header:
  seq: 2289276
  stamp:
    secs: 1679017616
    nsecs: 786542892
  frame_id: ''
sim_step: 2059654
identifier:
  data: "/ambf/env/BODY Base"
name:
  data: "Base"
wall_time: 2458.5556640625
sim_time: 2458.5556640625
mass: 0.0
pInertia:
  x: 0.0
  y: 0.0
  z: 0.0
pose:
  position:
    x: -0.7701
    y: -1.389849544726911e-20
    z: -1.0189
  orientation:
    x: 0.0
    y: 0.0
    z: -0.7071080798594737
    w: 0.7071054825112363
twist:
  linear:
    x: 0.0
    y: 0.0
    z: 0.0
  angular:
    x: 0.0
    y: 0.0
    z: 0.0
wrench:
  force:
    x: 5.053735208093713e-13
    y: 1.4175327578413999e-12
    z: -138.10000000000093
  torque:
    x: 0.6270453231448336
    y: 0.014909208154016529
    z: 0.052490783900325244
children_names:
  - Carriage1
  - Carriage2
  - Carriage3
  - RollArmBase
  - Endoscope35degreeinREMS
  - TiltProximalLinkage
joint_names:
  - carriage1_joint
  - carriage2_joint
  - carriage3_joint
  - roll_joint
  - TiltDistalLinkageandForceSensor-Endoscope35degreeinREMS
  - tilt_joint
joint_positions: [-2.1894120436627418e-06, -2.9813281798851676e-06, -3.4194874842796708e-06, -1.0471971035003662, 0.0, 0.0026905564591288567]
joint_velocities: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
joint_efforts: [-46.23480987548828, -45.937137603759766, -45.92805099487305, 0.01730331778526306, 0.0, 0.10672816634178162]
---
```

Note the input between running State and Command changed because the inputs of the controller were played with. It is okay that the output from State and Command don't agree here in the images.

An example of the joint could be Carriage1. run

rostopic echo /ambf/env/Carriage1/State

```
robotics@ubuntu:~/ambf/ambf_ros_modules/examples/object_control_example$ rostopic echo /ambf/env/Carriage1/State
header:
  seq: 137187
  stamp:
    secs: 1679017905
    nsecs: 273927450
  frame_id: ''
sim_step: 2258039
identifier:
  data: "/ambf/env/BODY Carriage1"
name:
  data: "Carriage1"
wall_time: 2747.041015625
sim_time: 2747.041015625
mass: 10.0
pInertia:
  x: 0.040501722882969465
  y: 0.041315055514847744
  z: 0.008895257795467525
pose:
  position:
    x: -0.7301000000002652
    y: -1.4692820415472714e-07
    z: -0.8522021894119765
  orientation:
    x: -3.0988623660401316e-14
    y: 3.0988436380129105e-14
    z: -0.7071080798594738
    w: 0.7071054825112362
twist:
  linear:
    x: -2.2204463285799952e-14
    y: 8.405335006702794e-17
    z: -5.291839015597599e-14
  angular:
    x: -3.030865431828495e-21
    y: 3.7417684040626e-20
    z: 8.881789156531812e-14
wrench:
  force:
    x: -0.0028016674171662255
    y: 4.552326432101946
    z: 46.234810584361206
  torque:
    x: 0.4644322223455989
    y: 0.07017089219106144
    z: -0.012231307450940376
children_names: []
joint_names: []
joint_positions: []
joint_velocities: []
joint_efforts: []
```

Publishing the state topics will give you specific information about the state of the robot and its joints.

# 4. AMBF and Blender for Editing ADF Files

If you want to change the robot ADF file, you can use the blender addon for AMBF. Follow the instructions here to install blender and the addon if needed.

https://github.com/WPI-AIM/ambf_addon

We can use Blender to update/change the robot dynamics, joint limits, saturation limits, etc. Navigate to the blender executable and run it by running (depending on your OS):

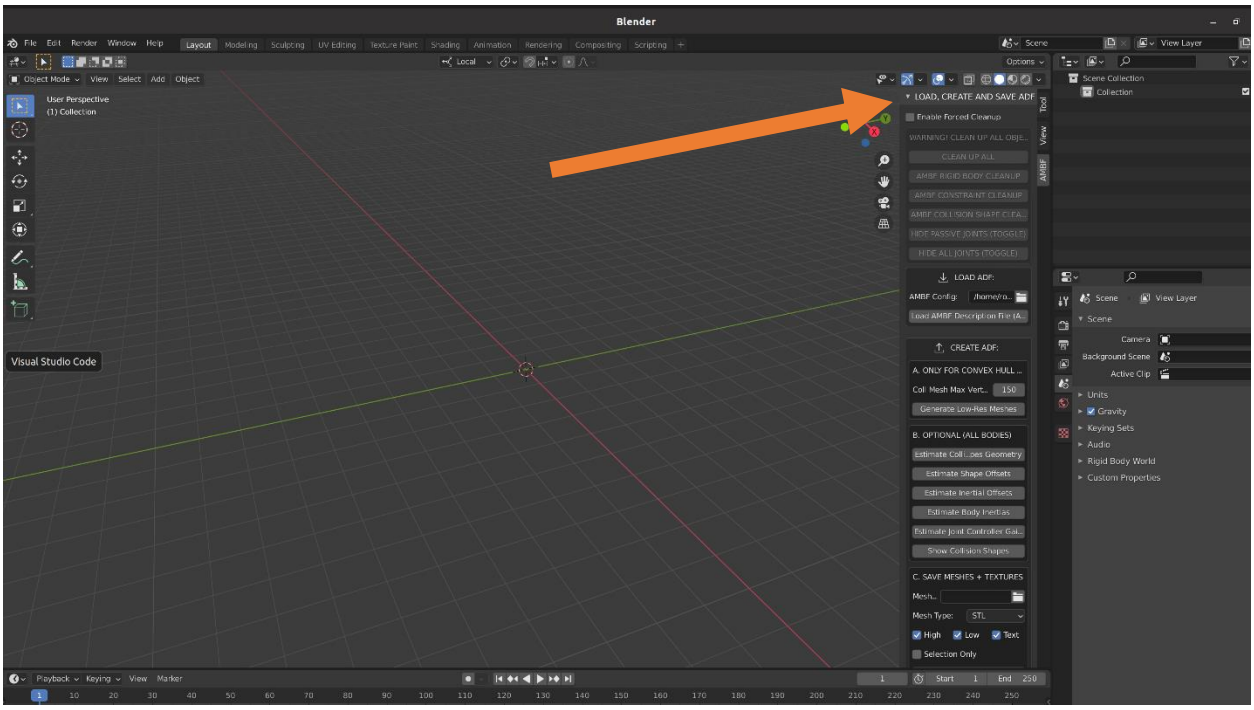
```
cd Downloads/blender-2.92.0-linux64/
```

and then run:

```
./blender
```

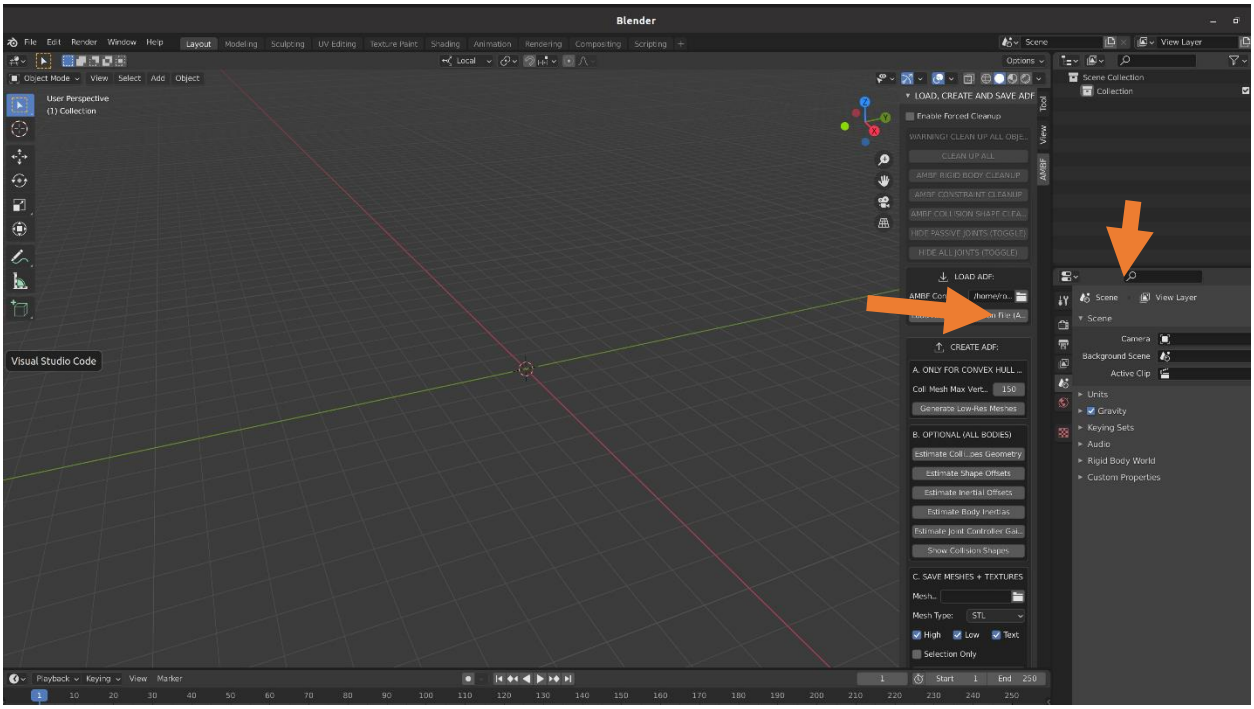
```
robotics@ubuntu:~$ cd Downloads/blender-2.92.0-linux64/
robotics@ubuntu:~/Downloads/blender-2.92.0-linux64$ ./blender
Warning: Unable to read '/home/robotics/.config/blender/2.92/config/userpref.blend': insufficient content
Warning: Could not find a matching GPU name. Things may not behave as expected.
Detected OpenGL configuration:
Vendor: VMware, Inc.
Renderer: SVGA3D; build: RELEASE;  LLVM;
Info: Deleted 3 object(s)
```

In blender you can press N or pull the side tab to open the AMBF addon window

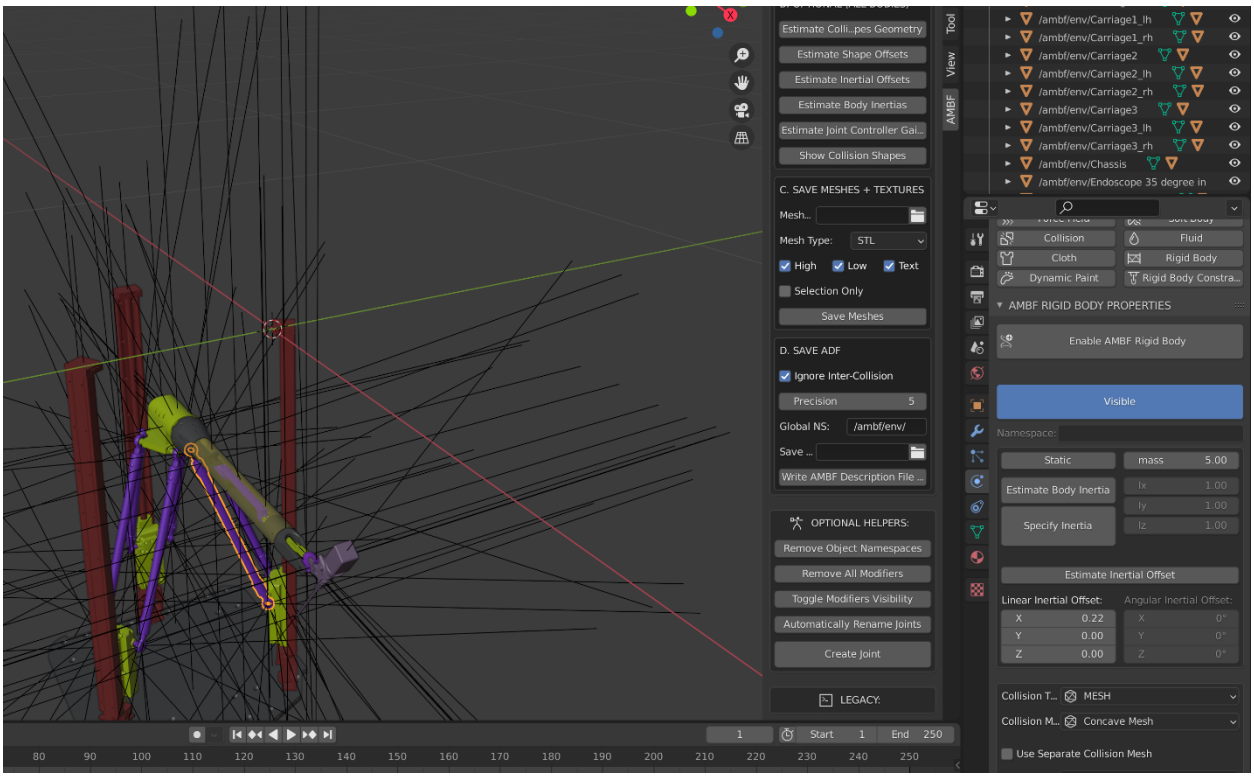



We can then load in the robot ADF by navigating to in our file explorer and then selecting the load AMBF Description file button. The Galen Robot should be located at

/home/robotics/Galen_AMBF_Simulation/ADF/galen.yaml




With the model loaded the joints and STL's can be observed. If you want to make changes to collision groups, damping, mass, system parameters you can play with the settings throughout the tabs. In general, most of the basic updatable parameters are found in the physics properties tab.



It is highly recommended to review the YouTube tutorials on the AMBF blender addon for updating ADF files. Here you can find the first of 4 short videos that explain the process to update and customize ADF files.

https://www.youtube.com/watch?v=iezOjg8qjZA&list=PLKH7Q-IzaPumDw77qQzF8deR1l4LgbWeP&index=3&ab_channel=AdnanMunawar