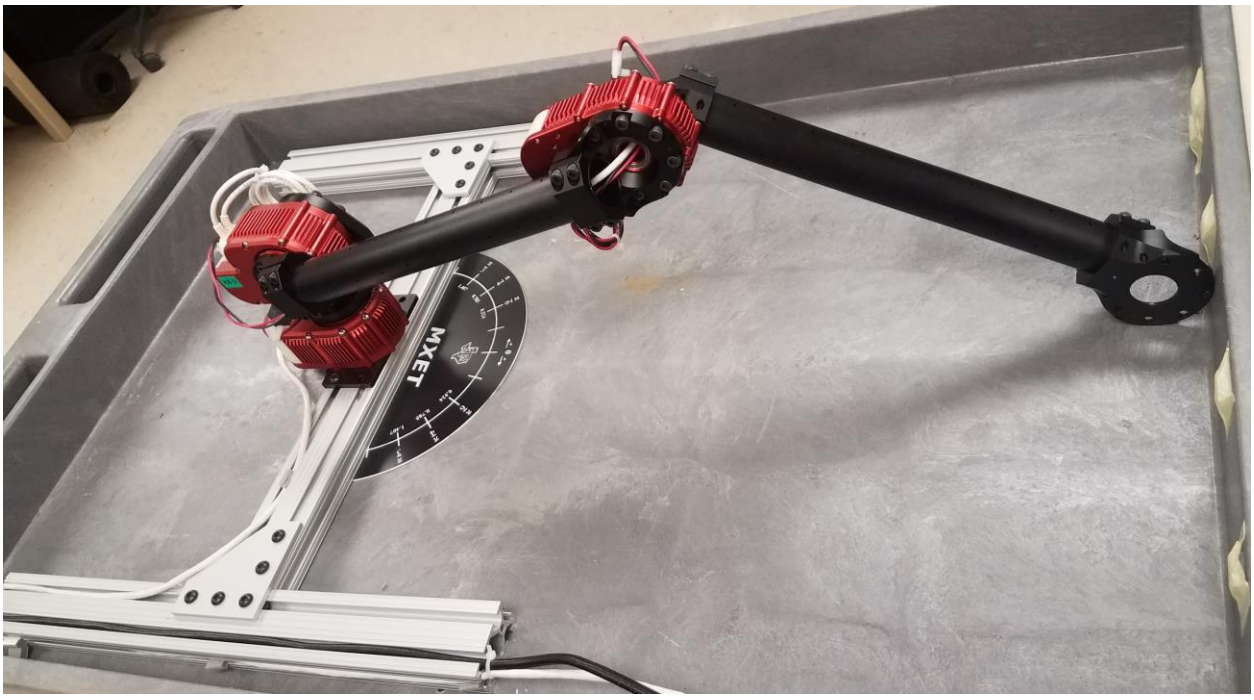
TEXAS A&M UNIVERSITY

# Hebi Robotics User Guide

Authors: Jeffrey Watts, David Malawey



Gather Documentation – Resource Summary 4
Hardware Documentation: 4
CAD files 4
MATLAB API 4
MATLAB help file 5
GitHub HEBI Directory 5
GitHub HEBI Matlab Exaples 5
Hardware Setup 5
Connect the Actuators 6
This should be done the first time you operate the robot on a new device. The kinetic information should be updated any time that changes are made to the robot itself. 6
Connecting to the actuators 6
setup hebi robot configuration/Kinematic Frame 8
Basic Functions 8
First Testing Example 9
Recording positions over time 9
Recording velocity over time 9
Examples 10
HEBI Scope- Open This First 10
Basic First Step of Running in Matlab 10
Example 1 HebiLookup 10
Example 2 getNextFeedback 11
Example 3 CommandStruct 11
Example 4 Kinematics 11
Example 5 Inverse Kinematics 12
Example 6 Trajectory Generation and Execution 12
Files 12
Waypoint Acquisition 12
Waypoint Movement and Simulation, Use ExperimentalDataAcquisition.m 13
Non-BlockingWaypointMovement 13
Inverse Simulation 13
DMTest (DMTest.m) 14

range_attemp_move.m 14
gravityTest.m 14
GravityFreeMode2.m 14
Simulink 15
How to Use: 15
Imported Data: 16
Physical Assembly: 16
Position: 16
Torque: 16
Power: 16
Known Facts about Actuators 18
HEBI Resources 21
Hardware Documentation: 21
CAD files 21
MATLAB API 21
MATLAB help file 21
GitHub HEBI Directory 21
GitHub HEBI Matlab Examples 21
Movement Sets 23


# Gather Documentation – Resource Summary

This section contains all of the downloads that are necessary for basic operation of the arm. Starting point for finding documents is at this location

It is advised that you read through HEBI's documentation in addition to this in order to gain a more complete understanding of the system. Specifically the getting started section should be read prior to doing the hardware setup or attempting to communicate with an actuator. The core concepts section should be read through before going beyond example 3.

Or, navigate by: HEBI website – software – API Matlab – link to docs wiki

- Software downloads
  - Get scope (GUI)
  - Get matlab API zip folder
- MATLAB API zip:
  - Extract to matlab workspace folder, then add to path using the addpath('filepath') command. The file-path can be obtained by clicking on that bar above where the files are displayed in most file explorers.

## Hardware Documentation:

http://docs.hebi.us/hardware.html

- "This section contains information specific to different hardware products and accessories. This includes information on modules and actuators, mechanical and electical accessories, robot kits, and details of any hardware-specific parameters for the APIs."
- Go to Hardware → Kinematic Information
  - Defines Twist, Extension, parameters used in matlab
  - Defines light Bracket, Heavy Bracket, left and right mounting

## CAD files

http://cad.hebi.us

- Links to GrabCad website with models of each actuator and bracket
- Includes 2d drawings of connectors and cabling
- Assembly drawings in pdf of robot assemblies such as "4dof arm"

## MATLAB API

http://docs.hebi.us/tools.html#matlab-api

- Gives an overveiw of how to configure and setup the HEBI using MATLAB.
- Contains links to specific commands, which contain more details on operation

### MATLAB help file

http://docs.hebi.us/docs/matlab/hebi-matlab-1.3-rev2169/

- Gives the details on using commands in MATLAB

### GitHub HEBI Directory

https://github.com/HebiRobotics

- Contains several HEBI programming examples, including
  - ROS
  - Matlab
  - C++

### GitHub HEBI Matlab Exaples

https://github.com/HebiRobotics/hebi-matlab-examples

- A subdirectory of the one listed above. Contains the following examples
  - Ex1_lookup
    - Starting point of the API, discovering devices, establishing com
  - Ex2_feedback
    - Accessing basic feedback
  - Ex3_command
    - Send basic commands, open-loop/closed-loop control, combined position/velocity control
  - Ex4_kinematics
    - Creating kinematic structures, fwd kinematics, jacobians
  - Ex5_inverse_kinematics
    - Inverse kinematics
  - Ex6_trajectories
    - Trajectory generation
- Also includes a couple of demos and predefined robot kits

## Hardware Setup

These two lists describe the physical setup of connecting to the robot as well as the physical setup of the robot itself. **Make sure the power is off before doing any of the following.**

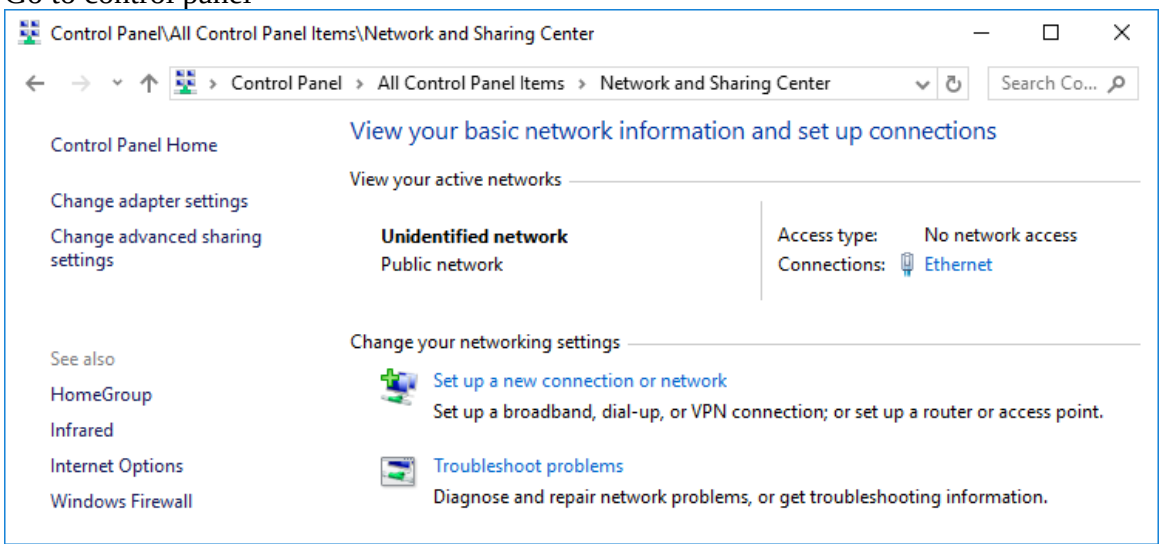
- D-link Model #: EBR-2310
  - PC connects to LAN-1
  - Actuator connects to LAN-2
  - Ensure that the actuator power supply is disconnected from 120v before connecting the DC output to the actuator
- Robot configuration as of 2018/May
  - X8-3: is mounted to chassis
  - X8-9: is mounted to X8-3 with a right heavy bracket in a right-inside configuration as defined here.

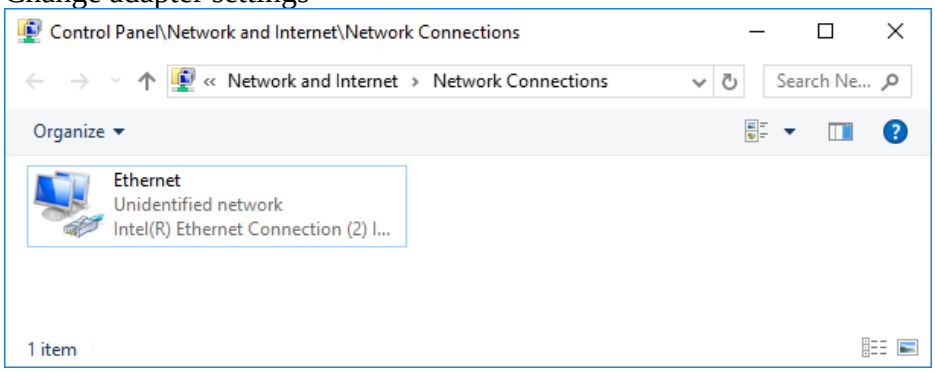
- X5-4: is mounted to the X8-9 with a X5 link at a 3.14 radian twist.
- Another X5 link is mounted to the X5-4 with a 3.14 radian twist.
- Ethernet cables are daisy chained and either port can be used as I/O

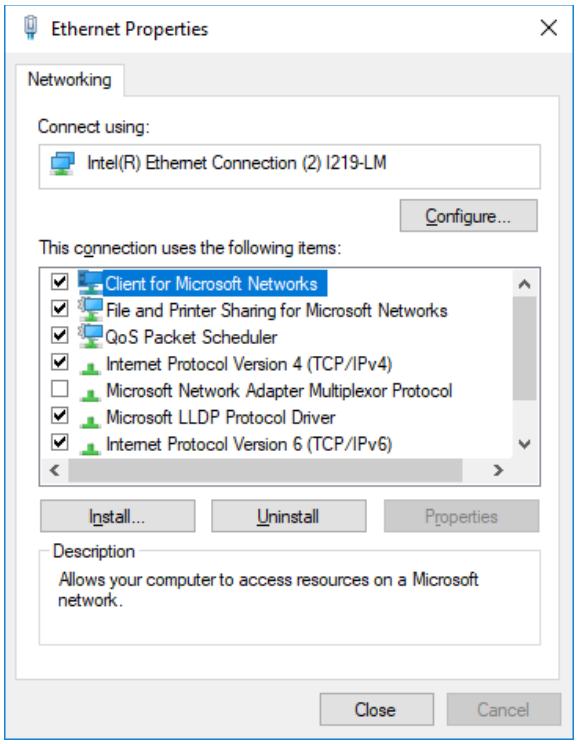
## Connect the Actuators

*THIS SHOULD BE DONE THE FIRST TIME YOU OPERATE THE ROBOT ON A NEW DEVICE. THE KINETIC INFORMATION SHOULD BE UPDATED ANY TIME THAT CHANGES ARE MADE TO THE ROBOT ITSELF.*

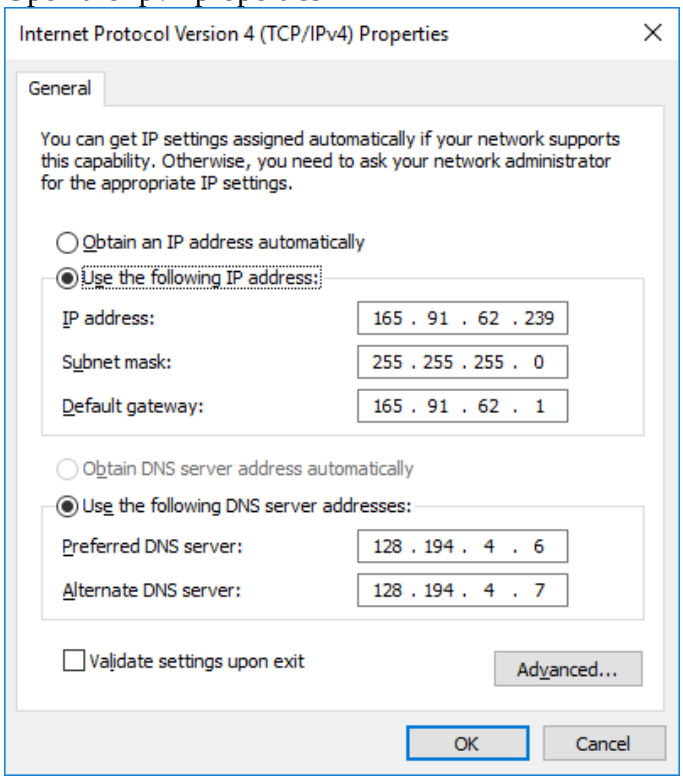
*CONNECTING TO THE ACTUATORS*

1. Open Scope.exe
2. Make sure your PC will automatically assign IP address. Note it is assumed you are using windows for this bit.
   a. Go to control panel
   b. 

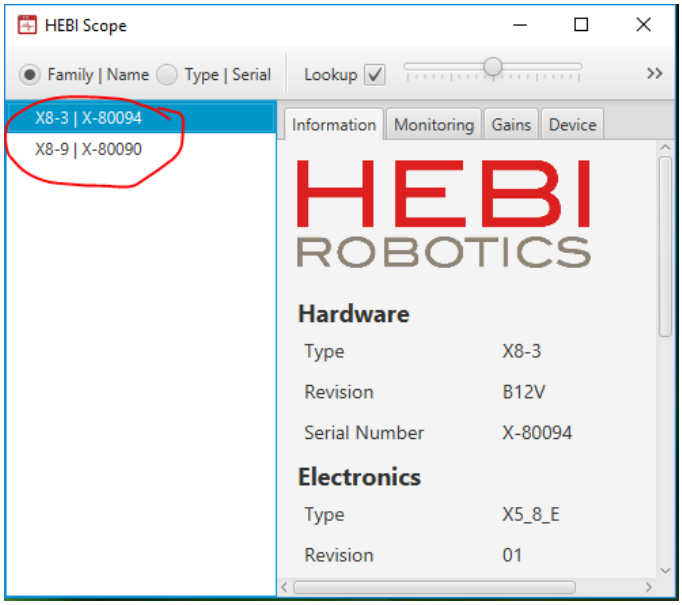
   c. Change adapter settings
   d. 

e. 


f. Open the Ipv4 properties

g. 


h. Change to automatic obtain IP address

i. 


j. Verify the actuators are recognized inside SCOPE.

*SETUP HEBI ROBOT CONFIGURATION/KINEMATIC FRAME*

This is handled by the KinematicSetup function which follows this process.

1. Create new .m file or use the provided function from the drive.
2. Setup Physical configuration of the robot.
    a. Use the table in the kinematics section of hardware to update parameters correctly. Found here. The general form of a kinematic setup can be found here.
        i. Extension = length of link
        ii. Twist = rotation about link axis
        iii. Mass can be input but it is already stored in program for default HEBI links, this should be checked and verified, our X5 links were last measured at .32kg.
        iv. Note: (which extensions and masses are OK by default)

## Basic Functions

These functions are required before running certain examples. Explanations are provided here because they were not found in the HEBI documentation or the function was created for our use.

**KinematicSetup**: This is used to generate the kinematic frame for the program and is effectively used as a neatness measure in the code to prevent clutter.

To create a kinematicSetup for a new configuration of robot, download this m file and follow the comments to customize it to your new config. Some information concerning actuator orientation can be found in the HEBI documentation here, we have also compiled some acceleration data for reference in the known facts about actuators section of this document.

**group.startLog()** : This is used to begin logging data during operation. The data it acquires is the same as successive calls to getNextFeedback would produce in addition to a time array. The logging is ended

with group.stopLogFull('LogFormat' , 'mat'); mat can be replaced with xlsx or csv. I generally use mat so analysis of the log can be quickly accomplished within matlab and no additional programs are necessary.

**Startup()** : This is used to trigger the hebi_load function which should load most of the HEBI files which in turn allow everything else to happen. In order for this to work make sure that the HEBI file containing the config files is either in the working folder or has been added through the addpath function. another way to do this is to check the current folder on the left side of the matlab window and if the file is listed but grayed out it can be clicked on and then added to path and selected folders and subfolders can be used. The addpath function is very important as without it verifying that you have access to all the various files and functions nothing will run properly. The nice part of that is if things haven't been added to the path matlab will generally throw an error indicating that the desired folder, function or file is not found in the current directory.

# First Testing Example

## Recording positions over time

```
% Display position feedback once a second for X seconds
group.setFeedbackFrequency(1); % reduce rate from 100 Hz default
t0 = tic();
i=1;
afbk = zeros(5,3); %create empty matrix, 5 rows, 3 col
while toc(t0) < 5
    fbk = group.getNextFeedback();
    disp(['position: ' num2str(fbk.position)]);
    afbk(i,:) = [fbk.position];
    i=i+1;
end
```

The above code is our example to record the positions of all actuators every second for 5 seconds. It comes from the example code found at hebi-matlab-examples/basic/ex2_feedback.m under the section "continuously access feedback." We have added a matrix "afbk" which stores the position values in each row. This does require group to be defined before calling this section of code.

## Recording velocity over time

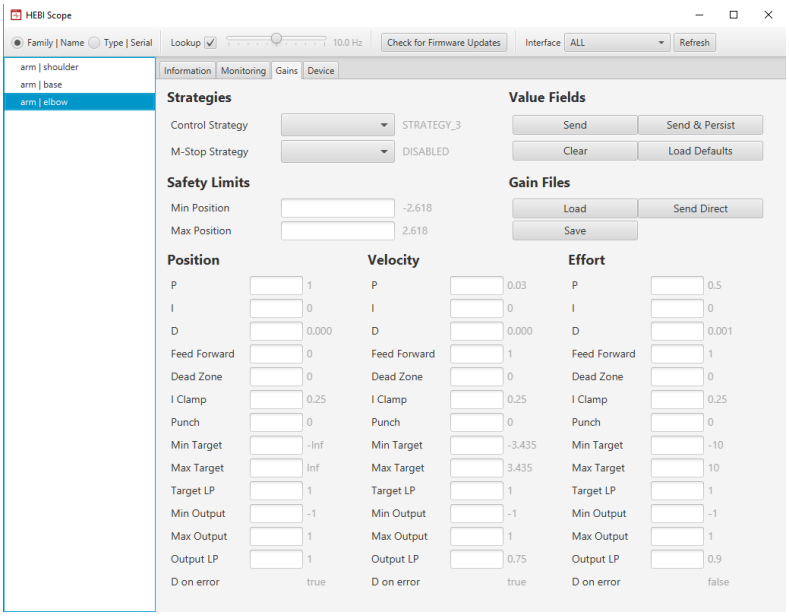
You can use the same code as above, but change the fbk.position to fbk.velocity

# Examples

If there is any confusion of an individual function that is provided by HEBI or the way they interact one excellent resource beyond the Matlab API section of their documentation is the Matlab help files which are found here. This information is also contained within the .m files that hebi_load loads in.

## HEBI Scope- Open This First

The HEBI scope should be the first thing you open. It allows you to change the family and name designations as well as the limits. The limits(in radians) should be set to reflect where wires or hard fixtures are in and around the robot in order to prevent hitting the table or twisting wires. PID settings can also be changed here and will eventually need to be looked at to optimize performance. If firmware updates are listed download and install those. Outside of the configuration settings scope allows the easiest commanding of the robot itself, one thing to be aware and careful of is that it is really easy to move extremely quickly using the manual controls which in turn allows or potential issues with hitting things or excessive jolting. The official HEBI documentation that goes into greater detail and allows for access to even more detailed descriptions by function, this can be found here. More detail on how to use scope can be found here.




The examples can be found here.

## Basic First Step of Running in Matlab

Use addpath to make certain that the hebi files have been added to the active folders list.

## Example 1 HebiLookup

**startup(); and KinematicSetup(); must be either run or in the code to allow this example to function. startup(); runs HebiLoad, which loads all the files Hebi needs to perform any task.** HebiLookup is used to find the device names and form a group for execution. ex1_lookup.m This link leads to the example code for how to use HebiLookup. There are a couple different ways to add the actuators to the MATLAB

API; creating a family, group, & names is one method, another is by serial numbers of the actuators. Below is example code from the HebiLookup example for adding the actuators using the family and group method. The family & names may be changed to any desired name inside of scope.

```
% Select devices by their user-settable family and name
family = 'arm';
names = {'base', 'shoulder', 'elbow'};
group = HebiLookup.newGroupFromNames(family, names);
display(group);
```

The code below will return all family & names of all actuators that have been added to the MATLAB API.

```
% Create a group of all modules on the network
group = HebiLookup.newGroupFromFamily('*');
info = group.getInfo();
display(info);
```

disp(HebiLookup); This will also return information on all actuators that have been added.

## Example 2 getNextFeedback

**startup(); and KinematicSetup(); must be either run or in the code to allow this example to function. startup(); runs HebiLoad, which loads all the files Hebi needs to perform any task.** getNextFeedback is used to obtain the current status information from each actuator. This information includes position, velocity, acceleration, torque, motor current and more. This feedback can be used to react to the current situation. ex2_feedback.m This link leads to the example code for how to use getNextFeedback. The following code will return the most recent feedback of all connected and group defined actuators.

```
% Display the most recent feedback
fbk = group.getNextFeedback();
display(fbk);
```

## Example 3 CommandStruct

**startup(); and KinematicSetup(); must be either run or in the code to allow this example to function. startup(); runs HebiLoad, which loads all the files Hebi needs to perform any task.** CommandStruct is the structure used to pass individual commands to the actuators. It can pass position, velocity and torque(effort). This example showcases the use of this in loops to generate specific behavior in a single actuator. ex3_command.m This link leads to the example code for how to use CommandStruct. In this example there are 4 different types of commands that are available; Command zero-effort (open-loop), Virtual Spring (closed-loop), Sine Wave (open-loop), Step Input (open-loop). In general this is used for individual actuator command, it is also used when using non-blocking trajectory commands. The general form of commands is cmd = CommandStruct(); followed by cmd.position = # where number is some radian value.

## Example 4 Kinematics

**startup(); and KinematicSetup(); must be either run or in the code to allow this example to function. startup(); runs HebiLoad, which loads all the files Hebi needs to perform any task.** This example goes

over how to setup the kinematic frame and obtain 4x4 transforms describing the output of each body based on the radial positions of each actuator. ex4_kinematics.m This link leads to the example code for how to use kinematics. The kinematic frame is the basis of trajectory generation and the various compensation algorithms. The basic form of declaring the kinematic frame is kin = HebiKinematics(); followed by a series of addBody commands such as kin.addBody('X8-3'); **The world frame is defined by the first body added to the kinematic frame.**

### Example 5 Inverse Kinematics

**startup(); and KinematicSetup(); must be either run or in the code to allow this example to function. startup(); runs HebiLoad, which loads all the files Hebi needs to perform any task.** This example explains how to go from the xyz position of the end effector to the radial positions of each actuator. One thing to take note of is the initialangles parameter which is used as a seed for the optimization of position and should be used to prevent configurations which involve going through the table. ex5_inverse_kinematics.m This link leads to the example code for how to use inverse kinematics. The optimization algorithm will always produce a result even if it is a result which violates physical constraints and as such should be checked before execution and if incorrect corrected, generally by setting more proper seed positions. The general form of an inverse kinematic calculation is positions = kin.getInverseKinematics('xyz', [.25 .25 .25], 'initial', [pi/2 pi/4 pi/2]);.

### Example 6 Trajectory Generation and Execution

**startup(); and KinematicSetup(); must be either run or in the code to allow this example to function. startup(); runs HebiLoad, which loads all the files Hebi needs to perform any task.** This example goes over how to generate the trajectory information and commands for going from one configuration to another. One thing to note and be aware of is whether you are using the blocking or non-blocking command configurations. Blocking will not allow for reaction to outside stimuli and will simply execute the trajectory until it finishes or is unable to continue. Non-blocking allows for a reaction to outside stimuli such as running into something. ex6_trajectories.m This link leads to the example code for how to use trajectory generation and execution. The general form of a call to trajectory generation is traj = trajGen.newJointMove(positions, 'Duration', time); where positions is a set of two or more radial positions for each actuator and time is a number in seconds representing the time to execute the trajectory in a blocking situation, in a two point trajectory it has been observed to act as half the time to traverse the trajectory and the total time so simulating or plotting the trajectory before execution is advised to check this. Blocking execution is generally of the form trajGen.executeTrajectory(group, traj);

## Files

### Waypoint Acquisition

The best way to obtain waypoints is to use fbk = group.getNextFeedback(); followed by positions = fbk.position; or instead of a direct equals if it is not the first point use positions = [positions ; fbk.position]; which will add on to the positions array.

## Waypoint Movement and Simulation, Use ExperimentalDataAcquisition.m

**This could be considered more a class of files than a single file as both the simulation and execution side are being updated frequently.**

**startup(); and KinematicSetup(); must be either run or in the code to allow this example to function. startup(); runs HebiLoad, which loads all the files Hebi needs to perform any task.** The WaypointSimulation.m file is used to set xyz positions(cm) and initial angle configurations(radians) for the robot to move through and then generate the full configurations and trajectories through them as well as simulating that using simulink which allows the user to verify the path and ensure that it will not attempt to go through the table or be a ballerina. In the more updated versions there is a variable called split which will determine how much time is allocated between each waypoint. This should be updated for both the simulation and the movement program.

The WaypointMovement.m file should be used in conjunction with the simulation file for generating the endpoints and initial angle configurations, basically copy paste from simulation into the execution file. Currently it uses a blocking execution method which moves between two endpoints for each iteration of its main loop. The split variable defines half the time between each point.

**The most updated versions of waypoint simulation and movement are now called ExperimentalDataAcquisition for simulation and CollisionDetectionMaybe for actual movement. As of 6/15/2018 most movement methods are non-blocking to allow for collision detection.**

## Non-BlockingWaypointMovement

**startup(); and KinematicSetup(); must be either run or in the code to allow this example to function. startup(); runs HebiLoad, which loads all the files Hebi needs to perform any task.** This file uses non-blocking commands with waypoint movement to allow for reactions which assist in preventing jitter and excessive shaking. This currently uses 1/100th of a second interval between each command, this is set in the config file as command frequency and is reflected in the generation of t prior to the while loop. This has been adjusted to include effort sensing in order to react to collisions. The simulation file associated with this is ExperimentalDataAcquisition.m. It is recommended that getNextFeedback be used to obtain the initial positions and manually enter those into ExperimentalDataAcquisition so as to obtain the most accurate simulation and avoid errors in pathing. **It should be noted that collisions as we currently have them coded trigger gravity free mode and as such are not something that can be simulated to please do not rely on collisions to do anything other than stop running into something.**

## Inverse Simulation

**startup(); and KinematicSetup(); must be either run or in the code to allow this example to function. startup(); runs HebiLoad, which loads all the files Hebi needs to perform any task.** The InverseSimulation.m file can be used to process a previously generated log file and simulate its movement. To use this you open the log file and then run InverseSimulation which will generate the arrays to pass to simulink as well as running the simulink simulation. As a extra bonus it currently calculates an estimate of power used as well as the actual power used. The estimate is semi accurate and should not be used for any high value decisions.

## DMTest (DMTest.m)

**startup(); and KinematicSetup(); must be either run prior to execution or in the code to allow this example to function. startup(); runs HebiLoad, which loads all the files Hebi needs to perform any task.** This example was created to train the HEBI of specific positions. To do this, move the end effector to a desired location and highlight

```
%move robot to first position
fbk = group.getNextFeedback();
position1 = fbk.position;
```

and press F9. This stores the location as position1. Continue to do this method with all the desired locations, once all the positions have been stored, run the code. The HEBI will revisit all the stored positions and then end the program.

## range_attemp_move.m

**startup(); and KinematicSetup(); must be either run or in the code to allow this example to function. startup(); runs HebiLoad, which loads all the files Hebi needs to perform any task.** This example allows for direct manual control of each actuator. It is setup to be controlled by entering in radian values into each set target. In the comments of the code it denotes which target is associated with its corresponding actuator. You may either type in the decimal value of the desired position, or you may type "pi/*" where '*' can be any value. Be careful with this as there are not safety features within this code, entering 30pi will result in the actuator rotating as fast as it possibly can in order to reach that angle.

## gravityTest.m

**startup(); and KinematicSetup(); must be either run or in the code to allow this to function. startup(); runs HebiLoad, which loads all the files Hebi needs to perform any task.** GravityTest allows the robot to go into a gravity compensated mode which attempts to hold position wherever it is located. Current issues with this include the existence of a kind of preferred position that if it gets near it will drift toward, the orientation generally occurs when the arm is pointing upward. It should also be noted that gravity test works best when you pull or push the robot by the end effector not by the rest of the robot as that will cause the end effector to drift downward.

## GravityFreeMode2.m

**startup(); and KinematicSetup(); must be either run or in the code to allow this to function. startup(); runs HebiLoad, which loads all the files Hebi needs to perform any task.** This is a function which when called stores the current position of the robot and then goes into gravity free mode for the amount of time specified in the call. At the end of the time it attempts to return to where it began before ending the function. This is still experimental and has not been uploaded yet, current errors to work through are collisions occurring during the return phase which call the function again which allows function calls ad infinitum. The another main issue encountered is a manifestation and complication of the tendency to go to a certain position if placed near it which causes issues with returning. The final large issue to work through is that the trajectories generated to return to the original position carry the risk of violating the physical constraints of the system to achieve the desired configuration, this is the same issue as seen in standard trajectory movement with the complication of being unable to effectively simulate it as by its nature the collision and subsequent movement are unpredictable.

### TwoPointMove.m

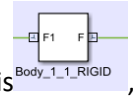
**startup(); and KinematicSetup(); must be either run or in the code to allow this to function. startup(); runs HebiLoad, which loads all the files Hebi needs to perform any task.** This is a function which is called by GravityFreeMode2 in order to generate and follow the return path. This is part of the issues found in return paths as well as its collision detection occasionally triggers during the movement due to too tight specifications.

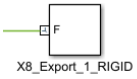
### CollisionToGravity.m

**startup(); and KinematicSetup(); must be either run or in the code to allow this to function. startup(); runs HebiLoad, which loads all the files Hebi needs to perform any task.** This is the newest and most experimental movement program which includes within it a call to GravityFreeMode2 when collisions are detected.

## Simulink

### How to Use:

First make sure you have a simulink license as without it you will be unable to import data from a workspace. The first and most effective test of this is to open the slx file in simulink, if an error is thrown there are two possible issues, the first is that you don't have a license and the second more likely option is that it cannot find some of its files to define the model. The first fix that should be done is to go to tools and enter the model explorer, click on the model workspace and change the data source type to a matlab file, following that click browse and locate the datafile, then relaunch simulink. The first file to check is is the datafile.m, verify that it is within the matlab current file. The second file to check is the assembly.xml file, verify that it is within the current file. The final set of files is the STEP files, if these are missing it should through an error but in the event that it does not you will be able to tell because when you run the simulink program it will show a blank screen instead of a moving arm. In order to fix this go to the rigid body blocks in the model, they look like this 

, once inside of that go to the definition blocks for that rigid body which look like this 

, note they each will have a different name which you should keep track of. Once inside of that open up the solid parameters block by double clicking on the block labelled Solid, in the Geometry section the third line should be a file-path, this needs to point to the location of the step file for that part so click on the path and navigate to the applicable part, following that hit apply and then F5, you should see the part appear in its model block. Do this for every single rigid body block and each rigid output definition in each block. That should allow you to then run the model and see the full arm. If after all of that, running the simulink file throws a very large block of red text try adding every file we have referenced here to the path. This should allow it to find the SMI data or whatever else it believes itself to be missing. If all else fails feel free to add the C drive to the path.

### Imported Data:

The model imports the position data named Motor1, Motor2, and Motor3 from the workspace and connects that to the input ports for each revolute joint.

### Physical Assembly:

The physical assembly was created using imported CAD files from the hebi online library. Each of these files was taken and made into sub assemblies. Each of the sub assemblies represents a different rigid body. These rigid bodies were then assembled with specific constraints. The constraints matched the physical motion of the robot. We then used a add-on in solidworks that transferred our assembly to simulink which converts the sldprt files to step files. The 'k' constant for each motor is the same as HEBI manufactures all of the X5 and X8 motors with 'k' values of 70. The 'b' value for each was obtained by plotting torque and deflection velocity and then normalizing that plot and finding an average.

### Position:

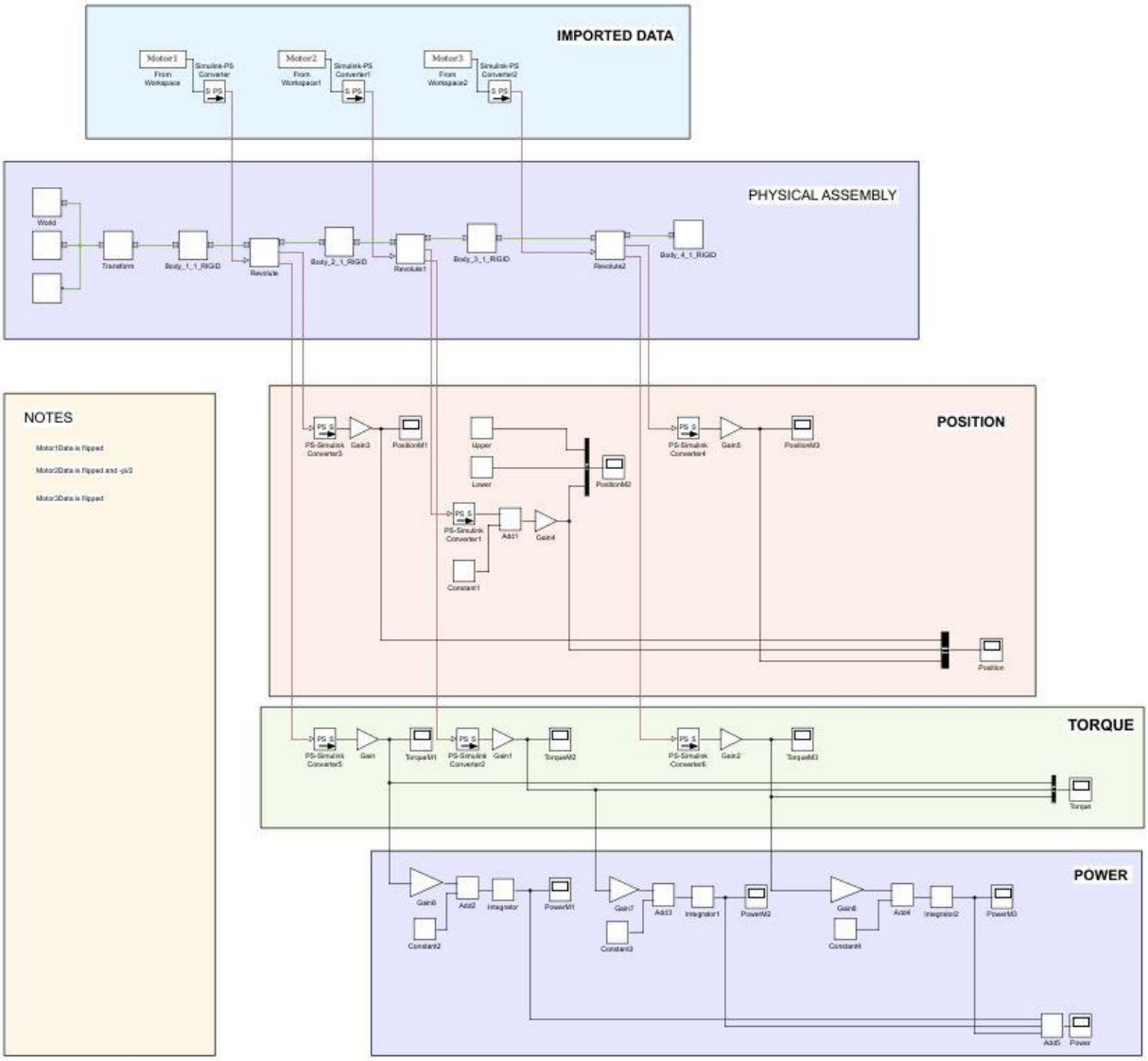
This section is for verification and visualization as it reverse conditions the position data coming out of the revolute joint so it matches the data fed to the actuators before feeding each position line to a scope for viewing. The scope for motor 2 includes two constants which represent the limits of the hardware's motion, if the position data ever is not between those limits, the initial angles and endpoint destinations need to be rethought.

### Torque:

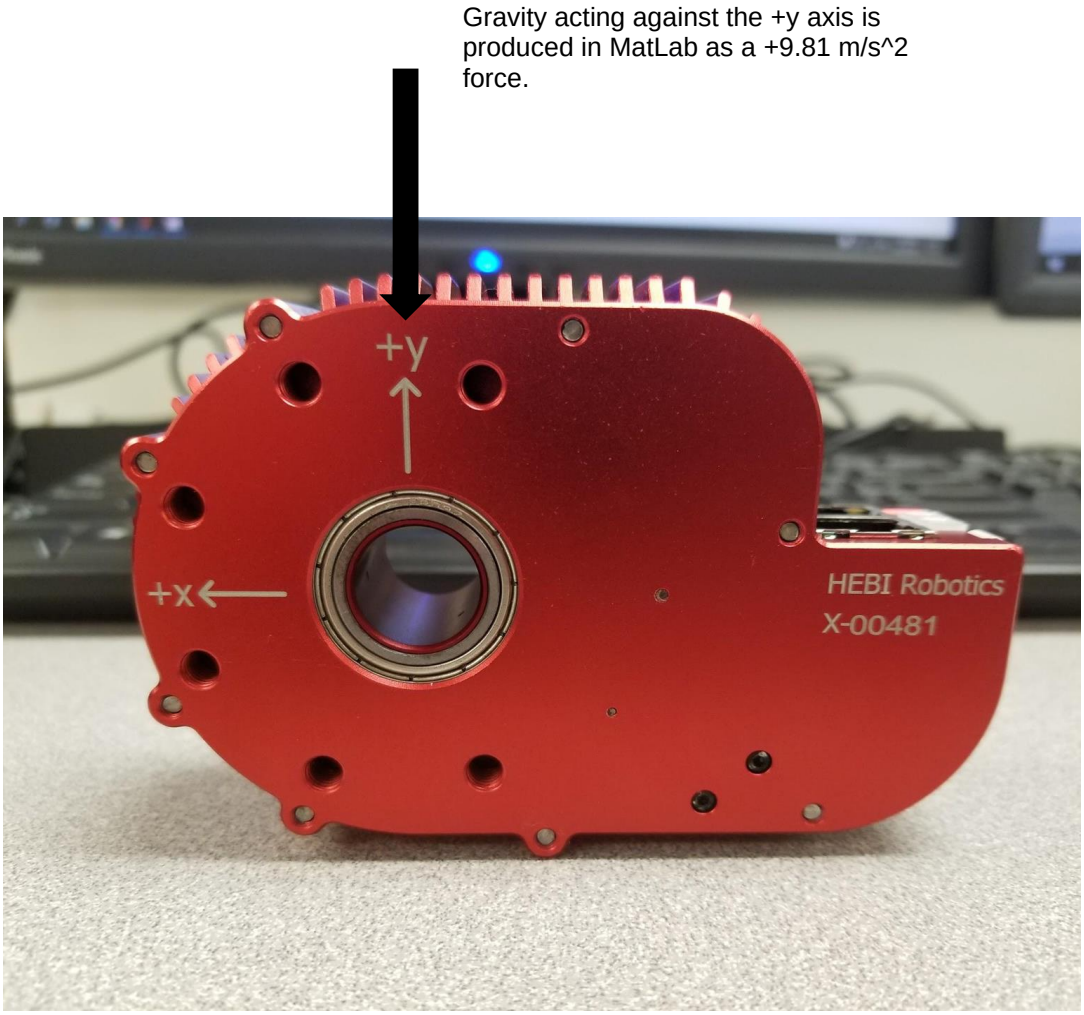
Each torque output from the model is automatically computed by simulink for each revolute joint. These are then conditioned to match in magnitude the torques produced by the physical motors. The reason for this conditioning is that the motors are controlled via PID loops which have gains ranging from .001 to .1 while the simulink model assumes a direct gain of 1 with no PID loop control when determining torque.

### Power:

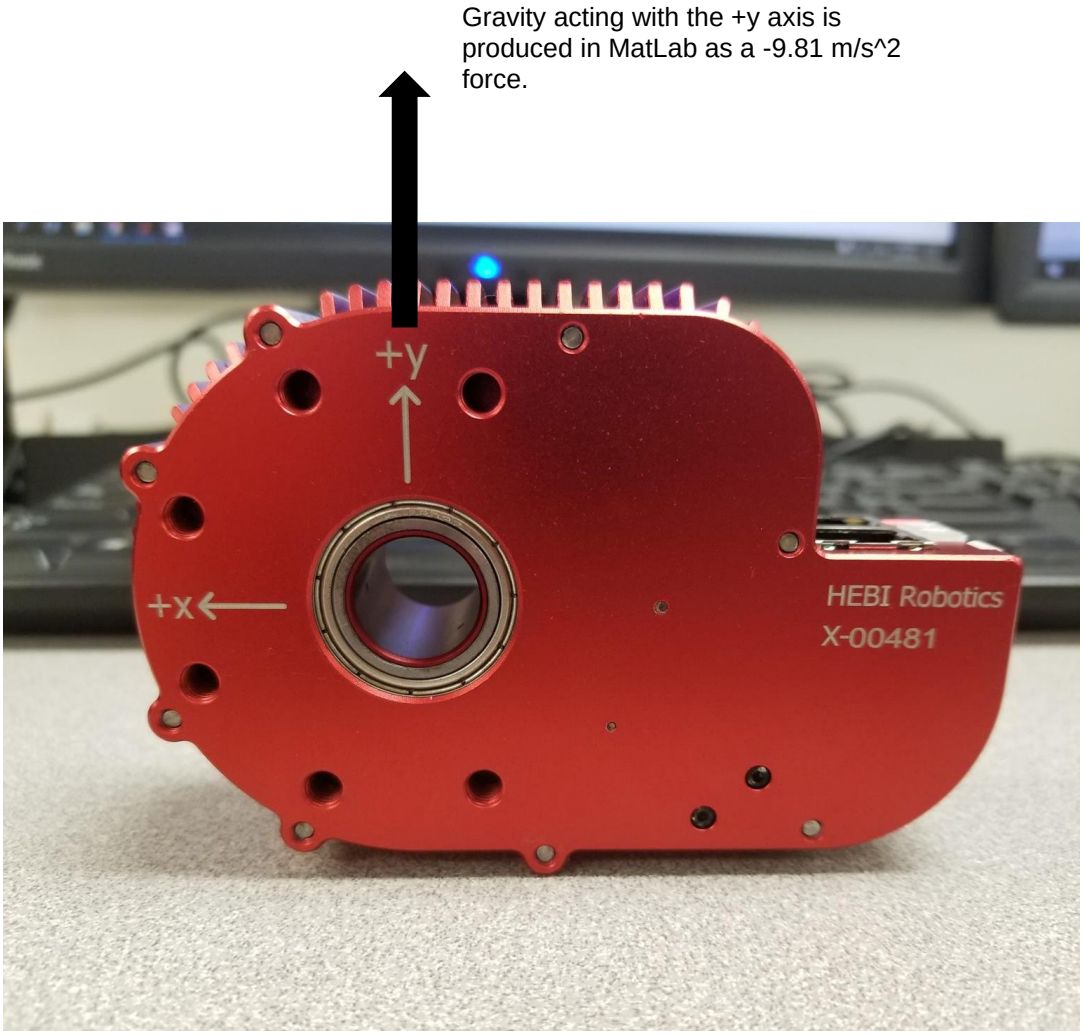
Each torque output after conditioning is run through a linear approximation of power as a function of torque and then wired to a scope for viewing. The torque to power equations are linear approximations of the cloud of data points produced from plotting torque to power and thus are prone to some error.


IMPORTED DATA
Motor1
From Workspace
Simulink-PS Converter
Motor2
From Workspace1
Simulink-PS Converter1
Motor3
From Workspace2
Simulink-PS Converter2
PHYSICAL ASSEMBLY
World
Transform
Body_1_1_RIGID
Revolute
Body_2_1_RIGID
Revolute1
Body_3_1_RIGID
Revolute2
Body_4_1_RIGID
NOTES
Motor1Data is flipped
Motor2Data is flipped and -pi/2
Motor3Data is flipped
POSITION
PS-Simulink Converter3
Gain3
PositionM1
Upper
Lower
PositionM2
PS-Simulink Converter4
Gain5
PositionM3
PS-Simulink Converter1
Add1
Gain4
Constant1
Position
TORQUE
PS-Simulink Converter5
Gain
TorqueM1
PS-Simulink Converter2
Gain1
TorqueM2
PS-Simulink Converter6
Gain2
TorqueM3
Torque
POWER
Gain6
Add2
Integrator
PowerM1
Gain7
Add3
Integrator1
PowerM2
Gain8
Add4
Integrator2
PowerM3
Constant2
Constant3
Constant4
Add5
Power

## Known Facts about Actuators




- Using the command “group.getNextFeedback()” produces all the feedback of the actuator
- `accelY: [-0.05844 9.835 0.2386] [m/s^2]` , this proves the statement in the figure above




- Likewise, gravitational force acting with the +y axis is described by MatLab as a -9.81 m/s^2
- We think this is HEBI Robotics way of describing gravity with the accelerometers within each actuator


- The figure above describes the direction of the +Z axis and the +Radian rotational direction
- The right hand rule applies and proves this case

## HEBI Resources

# HEBI Resources Summary

This section was compiled to understand from a high level where HEBI has placed its various instructions and documentation for all different aspects of the robot. This is useful to read entirely before browsing the HEBI documents to find some specific answers.

### Hardware Documentation:

http://docs.hebi.us/hardware.html

- "This section contains information specific to different hardware products and accessories. This includes information on modules and actuators, mechanical and electrical accessories, robot kits, and details of any hardware-specific parameters for the APIs."
- Go to Hardware → Kinematic Information
    - Defines Twist, Extension, parameters used in matlab
    - Defines light Bracket, Heavy Bracket, left and right mounting

### CAD files

http://cad.hebi.us

- Links to GrabCad website with models of each actuator and bracket
- Includes 2d drawings of connectors and cabling
- Assembly drawings in pdf of robot assemblies such as "4dof arm"

### MATLAB API

http://docs.hebi.us/tools.html#matlab-api

- Gives an overview of how to configure an
- d setup the HEBI using MATLAB.
- Contains links to specific commands, which contain more details on operation

### MATLAB help file

http://docs.hebi.us/docs/matlab/hebi-matlab-1.3-rev2169/

- Gives the details on using commands in MATLAB

### GitHub HEBI Directory

https://github.com/HebiRobotics

- Contains several HEBI programming examples, including
    - ROS
    - Matlab
    - C++

### GitHub HEBI Matlab Examples

https://github.com/HebiRobotics/hebi-matlab-examples

- A subdirectory of the one listed above. Contains the following examples
    - Ex1_lookup
        - Starting point of the API, discovering devices, establishing com
    - Ex2_feedback
        - Accessing basic feedback
    - Ex3_command
        - Send basic commands, open-loop/closed-loop control, combined position/velocity control
    - Ex4_kinematics
        - Creating kinematic structures, fwd kinematics, jacobians
    - Ex5_inverse_kinematics
        - Inverse kinematics
    - Ex6_trajectories
        - Trajectory generation
- Also includes a couple of demos and predefined robot kits

## Movement Sets

When generating these there are two methods that can be followed

1. +- x then y axis with return to start

```
start =[.0022 1.5591 .1205];
positions = start;
%       Move between a set of waypoints using blocking calls
xyz = [.25 0 .3];
initialAngles =  [0 pi/2 pi/4];  % or some pose that is close-ish to where you want to be
positions = [positions ; kin.getInverseKinematics('xyz', xyz, 'initial', initialAngles)];
xyz = [-.25 0 .3];
initialAngles =  [0 pi/2 -pi/4];  % or some pose that is close-ish to where you want to be
positions = [positions ; kin.getInverseKinematics('xyz', xyz, 'initial', initialAngles)];
positions = [positions ; start];
xyz = [0 .25 .3];
initialAngles =  [pi/4 pi/2 -pi/4];  % or some pose that is close-ish to where you want to be
positions = [positions ; kin.getInverseKinematics('xyz', xyz, 'initial', initialAngles)];
xyz = [0 -.25 .3];
initialAngles =  [pi/4 pi/2 pi/4];  % or some pose that is close-ish to where you want to be
positions = [positions ; kin.getInverseKinematics('xyz', xyz, 'initial', initialAngles)];
positions = [positions ; start];
```

**2**. 4 points above the table, rotation based.

```
start =[.0022 1.5591 .1205];
positions = start;
%       Move between a set of waypoints using blocking calls
positions = [positions ; [-.6195 1.0962 2.2179]];
positions = [positions ; [0.4547 0.8544 1.7493]];
positions = [positions ; start];
positions = [positions ; [-2.5258 1.0772 2.1851]];
positions = [positions ; [-3.7786 1.0838 2.1840]];
positions = [positions ; start];
```

**3**. 4 points above the table, flip based

```
start =[.0022 1.5591 .1205];
positions = start;


%       Move between a set of waypoints using blocking calls
xyz = [.2224 -.1487 .0967];
initialAngles =  [0 pi/2 pi/2];  % or some pose that is close-ish to where you want to be
positions = [positions ; kin.getInverseKinematics('xyz', xyz, 'initial', initialAngles)];
xyz = [.3422 .1762 .0923];
initialAngles =  [0 pi/2 pi/2];  % or some pose that is close-ish to where you want to be
positions = [positions ; kin.getInverseKinematics('xyz', xyz, 'initial', initialAngles)];
```

```
positions = [positions ; start];
xyz = [-.2208 -.1661 .0958];
initialAngles =  [0 pi/2 -pi/2];  % or some pose that is close-ish to where you want to be
positions = [positions ; kin.getInverseKinematics('xyz', xyz, 'initial', initialAngles)];
xyz = [-.2271 .1580 .0987];
initialAngles =  [0 pi/2 -pi/2];  % or some pose that is close-ish to where you want to be
positions = [positions ; kin.getInverseKinematics('xyz', xyz, 'initial', initialAngles)];
positions = [positions ; start];
```