ArmLab

# 

# Overview

## ACTING

* 6-DOF rigid-body coordinate transforms using homogeneous coordinate transforms
* Forward kinematics modeling of a manipulator
* Inverse kinematics modeling of a manipulator
* Grasping

## SENSING

* 3D image/workspace calibration
* Object detection with OpenCV
* Depth camera sensors

## REASONING

* Path planning & path smoothing
* State machines

**Your final project grade will consist of:**

* **Demonstrating checkpoints to the instructors before 5pm on Friday 10/4. (25%)**
* **A group lab report, due by 11:59pm on 10/11. (75%)**
* **Extra credit from Armlab competition on 10/7**

**This is a team assignment. Your team must complete it together, collectively participating on each component. Please review the course information and policy document for additional details.**

# Hardware

## RexArm

The RexArm base is a 6 DOF arm with 3D printed links and dynamixel servo motors. Three MX-28 form the base, shoulder, and elbow joints. Two AX-12 form the wrist. Currently the end effector is a stainless steel probe. This probe will be used while you build and test the teach-and-repeat and forward and inverse kinematics sections of your code. For the final competition, you will use the provided gripper or design and build a gripper using dynamixel XL-320 servo motors to allow the arm to pick up, move, and accurately place 1.5” colored blocks.

## Dynamixel Motors

|  |  |  |
| --- | --- | --- |
| MX-28    [Data Sheet](http://support.robotis.com/en/product/actuator/dynamixel/mx_series/mx-28at_ar.htm)  Voltage: 12V  Stall Torque: 2.5 N⋅m  Max speed: 55 RPM  Encoder: Magnetic (360°)  Resolution: 0.088° | AX-12    [Data Sheet](http://support.robotis.com/en/product/actuator/dynamixel/ax_series/dxl_ax_actuator.htm)  Voltage: 12V  Stall Torque: 1.5 N⋅m  Max speed: 59 RPM  Encoder: Resistive (300°)  Resolution: 0.29° | XL-320    [Data Sheet](http://support.robotis.com/en/product/actuator/dynamixel_x/xl_series/xl-320.htm)  Voltage: 7.5V  Stall Torque: 0.39 N⋅m  Max speed: 114 RPM  Encoder: Resistive (300°)  Resolution: 0.29° |

## Serial Interface & Power

Dynamixel servos use a three wire connection and are all wired up in parallel. The two connectors on either side of the motor are identical, and the order motors are connected does not matter. Pin 1 is the ground reference pin. Pin 2 is the power pin - 12V for the MX and AX series motors, and 7.5V for the XL series motors. Pin 3, the data pin carries information using a half-duplex asynchronous serial protocol, meaning transmit and receive data are carried on the same physical wire at a fixed (1Mbps) baud rate. Because the XL series motors require a lower power voltage, a voltage regulator board must be used between the first four joints of the Rexarm and your gripper

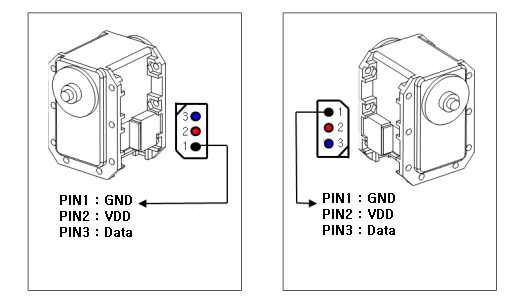


Figure 3. Dynamixel Connections

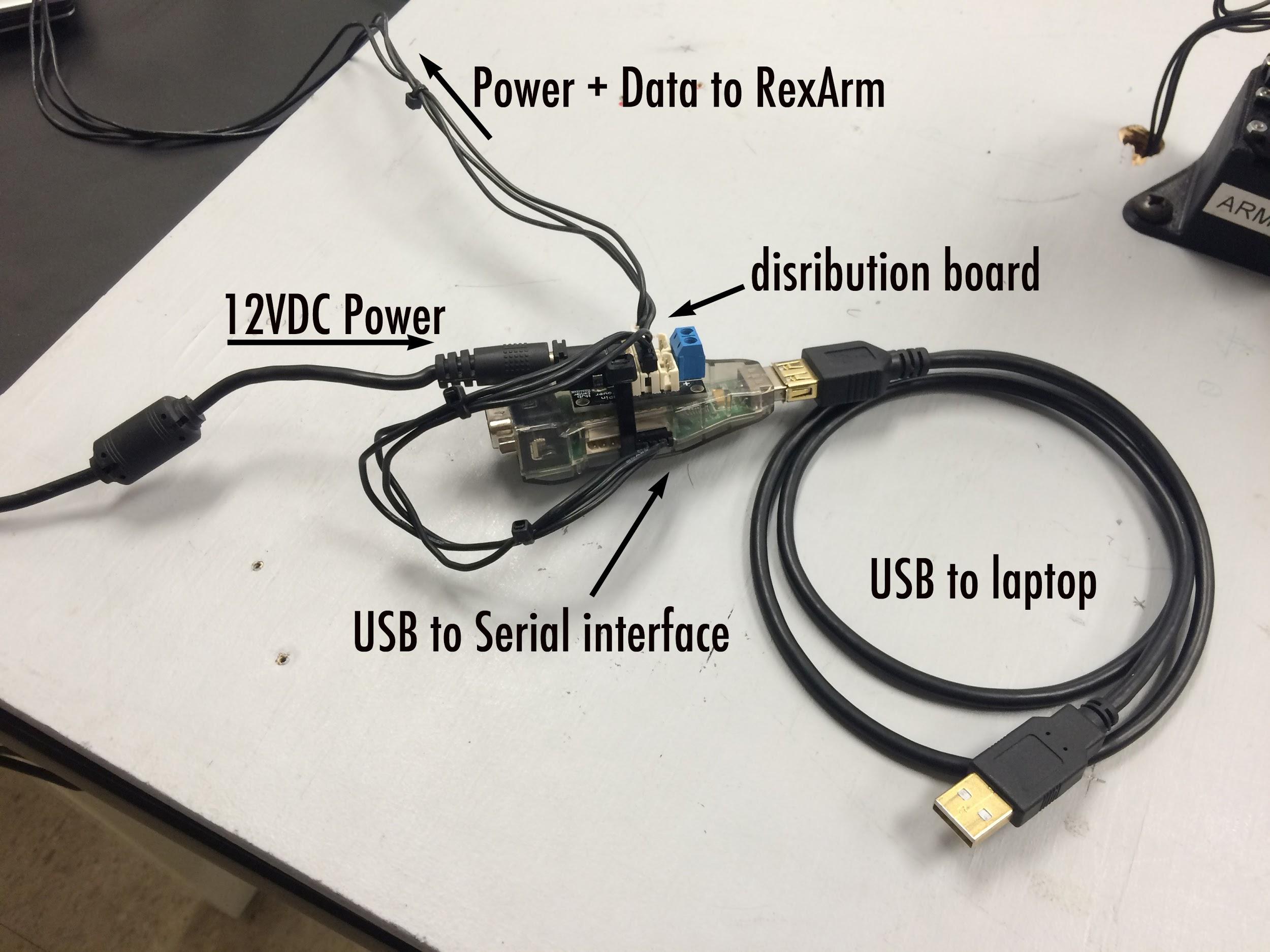


Figure 2. Power and data connections for the RexArm

Before you run the code, make sure all the connections are made to the RexArm. You will need a USB to serial interface (either a USB2Dynamixel or a USB2AX) with power distribution board and a 12VDC power supply connected as shown in Fig. 2. The RexArm in interfaced as a standard serial device in Linux. The larger Dynamixel2USB will show up in the filesystem at /dev/ttyUSB0, the smaller USB2AX will show up at /dev/ttyACM0. We have found that the larger Dynamixel2USB is slow and causes lag and jitter in the arm. We will be using the smaller USB2AX at /dev/ttyACM0.

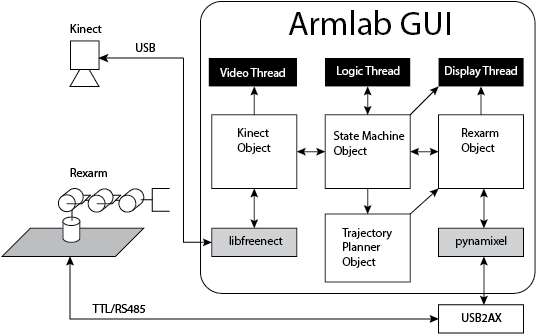
## 

## Kinect Sensor

The Kinect sensor requires a power supply and USB connection and consists of an RGB camera along with an IR laser projector and IR camera. The laser projector projects a pattern of invisible IR light onto whatever the camera sees. An IR image is recorded and the distortion of the pattern in the image enables the sensor to measure the distance to objects in the field of view. We will use the open source freenect driver to connect to the Kinect and capture images. To make sure the kinect is working, connect it to the lab laptop and run one of the freenect demo programs. A window should open showing both the RGB image and a grayscale image representing the depth measurements. You should note that the depth image is only a rough measurement of depth at each pixel, and has difficulty with surfaces that are not fairly smooth or are at an extreme angle relative to the optical axis of the camera.

|  |
| --- |
| $> freenect-cppview |

# Software



## 

## armlab-f19

All code files that should be modified for this lab are at the top level of the armlab repository. We provide a basic description of the given files below. Please check the comments inside each file for additional descriptions. The code given is just a minimum implementation if completed. You are welcome to add more files, classes, etc. and change the code organization in anyway you see fit.

|  |  |
| --- | --- |
| control\_station.py | Main program. Sets up threads and callback functions.  You will upgrade some functions and also implement others according to the comments given in the code. |
| pynamixel/ | This is a library of functions to interface with the official DynamixelSDK from Robotis using python. Files define the same functions for each type of joint. These joints are passed to the Rexarm class where member functions are provided to handle most relevant functionality. |
| rexarm.py | Implements the Rexarm class.  This class contains:  last joint command  last feedback from joints  functions to command the joints  functions to get feedback from joints  functions to do FK and IK  You will upgrade some functions and also implement others according to the comments given in the code. |
| kinect.py | Implements Kinect class.  This class contains:  functions to capture and convert frames  functions to load camera calibration data  functions to find and perform 2D transforms  functions to perform world->cam and cam->world transforms  functions to detect blocks in the depth and rgb frames  You will upgrade some functions and also implement others according to the comments given in the code. |
| state\_machine.py | Implements the StateMachine class  The state machine is the heart of the control  You will upgrade some functions and also implement others according to the comments given in the code. |
| trajectory.py | Implements the TrajectoryPlanner class  You will upgrade some functions and also implement others according to the comments given in the code. |
| kinematics.py | Implements functions for forward and inverse kinematics.  You will fill in the functions in this file and add new functions. |
| mainWindow.ui | This file defines the GUI interface, created using QtCreator.  No changes are needed. You can edit this using QtCreator. To compile a new ui.py file, run:  pyuic4 mainWindow.ui -o ui.py |
| ui.py | Output from QtCreator with GUI implementation in Python.  You can check the name of the GUI objects looking at this file.  Automatically generated, no changes are needed. |
| camera\_cal.py | Standalone program to generate camera distortion parameters and camera intrinsic matrix after calibrating with a checkerboard.  Outputs the camera calibration into the calibration.cfg file. You will need to modify code to read in and use the calibration as needed. |

## 

# 

# Setup

## Getting the code

Begin by forking the armlab repository on the ROB550 Gitlab using the web interface to a group on Gitlab made for just your team. Then you can clone your group repository to your local machine.

## Running the code

|  |
| --- |
| $> ./control\_station |

The first time you run the program it may fail, or the video may not show up. This is because the software that interfaces with the kinect is not configured correctly. to force it to have the correct configuration, run the freenect-cppview program and make sure the kinect is streaming both video and depth data.

|  |
| --- |
| $> freenect-cppview |

Once you can get everything running, take a look through the codebase and get started on the lab tasks.

# Lab Tasks

## 1. Basic Motion

This lab will be done with the Qt GUI interface already provided. Therefore, it is important for you to understand how it works and how to interface with it.

The control\_station program launches three threads. First the kinect is handled in its own thread, VideoThread, to capture and display the RGB and Depth images from the camera. The all important state machine is run in its own thread, LogicThread, and run at a fixed rate determined by the sleep time in the thread’s run() function. All of the messages to the rexarm are sent using the state machine for both commands and feedback. In idle mode (the initial state) the only commands sent are to receive feedback. The measured data are stored in the Rexarm object. A third thread, DisplayThread, updates all of the read outs in the GUI from data from the Rexarm and State Machine.

When you launch the program, you should put the arm in the “manual” state by clicking the manual check box above the grayed out sliders. This will switch states from idle to manual and let you control the individual motors with the sliders.

Now that you can control the arm, you will note that there are physical limitations for each joint. If you command a value greater than that which is physically achievable, it is still commanded to the servo. The servo will then drive up against the physical obstacle and could therefore be damaged. Thus, before you proceed, write a clamp function in rexarm.py to constrain the joint angles of the arm.

|  |
| --- |
| **1.1:** Use the control\_station.py app to determine minimum and maximum joint angles for each motor. You can set the torque to zero and then freely move the arm while feedback is given. Implement the clamp function in the Rexarm class that safely limits the joint angles by limiting the commanded joint angles in the class so that they are within its physical operating range. Test that the sliders do not allow you to command the arm past the physical limits |

Now you can safely send commands to the arm without accidentally sending a command that will harm the motors.

|  |
| --- |
| **1.2:** Create a new **execute** state and associated execute function in state\_machine.py. You should be able to transition to this state from **idle** and from this state to **estop**. When completed this state should transition back to to idle. In the function set up a simple waypoint follower that executes the plan below by using the rexarm.set\_positions() function and uses the rexarm.pause() function to wait a reasonable amount of time for each waypoint to be reached. rexarm.pause() is a special wait function that reads the feedback from the arm while waiting. Set up the execute button in the GUI to transition to the **execute** state. The configuration space waypoints the arm should follow are:  [[ 0.0, 0.0, 0.0, 0.0, 0.0],  [ 1.0, 0.8, 1.0, 0.5, 1.0],  [-1.0,-0.8,-1.0,-0.5, -1.0],  [-1.0, 0.8, 1.0, 0.5, 1.0],  [1.0, -0.8,-1.0,-0.5, -1.0],  [ 0.0, 0.0, 0.0, 0.0, 0.0]] |

At this point work can be divided among your team. You can do this in any way you like, but it is suggested that one member can take the lead on the OpenCV camera calibration and object detection, one member can take the lead implementing the teach and repeat and forward kinematics, and the final member should take the lead on the gripper design and 3D printing. All three group members are responsible for understanding all aspects of the project even if it is not their primary responsibility

## 

## 2. Teach & Repeat / State Machine

Teach and repeat is a very common way of programming robotic arms to do repetitive tasks. It allows the user to command the robot with no special skills. In this task the idea is to implement a system that takes a set of discrete waypoints in joint space and commands the arm to follow the path connecting them all. To get the waypoints, we will be teaching the Rexarm to “play” the board game Operation.

You will set the torques of the motors to zero while physically moving the arm around to different poses and recording the joint angles at each pose. Then, you should be able to press a button to execute the playback of this list of poses. The specifics of how you implement this is up to you, however, it should be implemented in the GUI in such a way that it is obvious for any user. You can use the status messages at the bottom of the GUI window to issue instructions.

You should not use the gripper you are designing, only the probe tip of the Rexarm. You can connect the red wire on the Operation board to the probe and it will buzz when the probe collides with the edge of the target areas.

After your simple waypoint follower is implemented you will implement a cubic or quintic polynomial scheme in trajectory\_planner.py to produce a well defined velocity profile for each joint. To do this you will need to control both the position and speed of the joints.

You can control the motors' speed and max torque individually. The speed and position can be controlled simultaneously in joint mode only as long as enough torque is available. The MX28 motor speed range is 0x000 to 0x3FF, and each bit corresponds to about 0.114 rpm (see: [here](http://support.robotis.com/en/product/dynamixel/mx_series/mx-28.htm)) The AX12 has the same range and and each bit corresponds to about 0.111 rpm (see: [here](http://support.robotis.com/en/product/dynamixel/ax_series/dxl_ax_actuator.htm).) The lcm messages from the rexarm driver have motor speed and load feedback data available, you can see all of the info by using lcm-spy while running the rexarm\_driver just as in the lcm tutorial.

|  |
| --- |
| **2.1:** Implement “teach-and-repeat” functionality as a set of buttons controlling the state machine. the arm is placed into idle mode (no torque) and the user then leads the arm through the workspace and records a series of waypoints in configuration space by pressing a “record waypoint” button in order to “play” the game Operation. You should use the user buttons to enable user friendly interaction with the process.Complete the state machine to repeat the motion of the arm through the recorded waypoints. Record the feedback of the joint positions to a file so you can later generate a plot with the waypoints and the path at low and high speed. CHECKPOINT:   * Demonstrate to an instructor the teach-and-repeat functionality |

You may notice at high speed the Rexarm does not move smoothly or gracefully. The next task in this section will be to implement a trajectory planner to limit the accelerations of the joints and generate smooth motions. We will not cover this until halfway through the lab so you may want to skip the next task and come back to it later.

|  |
| --- |
| **2.2:** After we cover trajectory generation in class, implement a Cubic or Quintic Polynomial scheme in trajectory\_planner.py described in Spong Sec. 5.5. Repeat the operation game and find a way to make the probe have zero velocity at each board location with intermediate points that have velocities you set for each individual motor depending on the speed of execution of the spline path. REQUIRED FOR REPORT:   * Describe how you implemented the path smoothing * Generate a plot of the robot end-effector position in world coordinates at slow and fast speed with and without path smoothing (4 curves) * Generate a plot of the magnitude of end-effector velocity with and without path-smoothing at high speed |

## 3. Gripper Design

This task asks you to design a gripper for the final competition. For the gripper design, you will be given three XL-320 motors, some standard Robotis OLLO brackets, a bag of Robotis OLLO rivets and an in-line voltage regulator/adaptor. You do not need to use all the motors in your gripper design if you feel you do not need to. You will also be given 3D models of the motors and standard brackets. The [Gripper Design](https://drive.google.com/open?id=1_yJGwBKaNwz39yToMga-Pv901MFjzIDOnQWytMwh8G0) document has more information on this task. A tutorial for designing and building a generic OLLO adaptor for the AX-12 motor is provided in the ROB550 tutorials folder. Your gripper design must be able to pick up the targets on the board placed between 60 mm and 300 mm from the center of the arm base. The gripper should be 3D printed and use OLLO rivets to connect the motors and different pieces together. It is a good idea to engineer some compliance into your gripping mechanism, or add additional gripping materials (like from cut up rubber gloves) to provide more grip with less grasping force as the XL-320 motors have limited torque.

You must design your gripper to have free access to at least one connector of one of the motors for quickly plugging and unplugging your gripper from the chain. XL-320s operate off of a lower voltage than the MX28s and AX12s so you will need to use the provided in-line voltage regulator to power them.

|  |
| --- |
| **3.1:** Using 3D CAD tools, design a gripper capable of manipulating blocks as required for the competition (see end of lab document). Further information is found in the Gripper Design Parameters document on the course Google Drive. REQUIRED FOR REPORT:   * Describe the design and design choices/tradeoffs you made. * Discuss any issues that became evident in the design and how you revised it. * Include drawings of your gripper assembly. |

|  |
| --- |
| **3.2**: Create STL files for your parts, 3D print them and assemble and test the gripper.  CHECKPOINT:   * Show your gripper to the instructors and discuss your design with them.   FOR REPORT:   * Submit an archive containing design files and STL files. |

## 4. Kinematics

|  |
| --- |
| **4.1**: Measure the link lengths using calipers. You must implement two forward kinematics functions that take 4 joint angles and returns the wrist location in workspace (global frame) coordinates. Your function will use the DH convention.The end effector position is defined by x,y,z and where the orientation uses the common ZYZ Euler angle convention. You should display the coordinates of the tip of the end effector in your GUI using the text labels provided in the upper left. REQUIRED FOR REPORT:   * Include a DH table for the Rexarm in your report * Describe how you verified that your arm tooltip achieved expected workspace coordinates for your test cases, estimate the error, and justify the accuracy of your measurements. |

|  |
| --- |
| **4.2**: Draw a diagram of the robot arm with all of the parameters used to calculate the inverse kinematics. Implement an inverse kinematics function that returns the joint angles that reach a position in the workspace. This function should return error messages if the specified tooltip workspace configuration is not reachable, and it should be able to handle degenerate poses. REQUIRED FOR REPORT:   * Include a schematic diagram of the arm with all relevant FK and IK parameters * Precisely specify the IK function by writing the equations, explaining them, and referencing the source file and lines |

## 5. Camera Calibration & Block Detection

For this part, you will be performing a camera calibration to get workspace coordinates from your camera. You will need to find the intrinsic matrix for the Kinect camera by using the provided camera calibration python program and provided checkerboards. This program will also find lens distortion parameters which you can use to transform the video to get a less distortion, though this is not necessarily needed as the Kinect image is relatively low distortion in the region you are interested in imaging, and the distortion for the depth camera is unknown.

A reasonably good function to relate the depth value and Z distance from camera is:

|  |
| --- |
| Z = 0.1236 \* np.tan(d/2842.5 + 1.1863) |

This function is a good approximation of depth over the entire range of the Kinect depth sensor. However you are only interested in finding blocks in a narrow volume of space, so it may be you get better results by finding your own depth calibration function.

|  |
| --- |
| **5.1:** Implement a workspace calibration procedure activated using one of the user buttons in the GUI in the state machine (create a calibration state). The procedure should set status messages at the bottom of the window to lead the user through the calibration.Use mouse clicks to get pixel locations of known locations in the workspace Repeat with the depth frame and use an affine transformation to register the two together.Using the intrinsic matrix you find for the RGB camera and the depth calibration function, create a function that takes pixel coordinates from the image and returns workspace coordinates.Update the display under the video in the GUI to show workspace coordinates of the mouse location as you hover over the video. Measurements only need to be valid for points on or above the board. *Suggestion: A reasonable camera extrinsic matrix can be found clicking on known locations in the camera image. You could make this more robust, or even automatic by attaching some calibrated object to the end effector and using your FK to find correspondences between points in the images of the calibration object and the Rexarms coordinate frame.*  CHECKPOINT:   * Demonstrate to an instructor the calibration procedure   REQUIRED FOR REPORT:   * Report your intrinsic matrix and depth calibration function. * Describe how you verified the calibration was correct and provide evidence for the accuracy of your calibration |

## 

|  |
| --- |
| **5.2:** Implement a block detector that is capable of detecting black, red, orange, yellow, green blue, purple, and pink blocks and determining their location in the workspace. There are many valid approaches to this task but it is much more reliable to detect blocks in the depth image, and then figure out what color they are. Example images are available on the Google drive. It is recommended that you create a standalone program first following the OpenCV and python tutorial, and then implement these functions in the Kinect class in the kinect.py file. REQUIRED FOR REPORT:   * Describe the methods you used to implement the block detection and any strategies you implemented to limit false positive detections and enhance the robustness of the detector * Describe how you verified the accuracy of your block detector and provide evidence of its performance. Create a plot that visualizes the uncertainty vs. location on the board. |

## 6. Final Bits & Pieces

Now you need to put all the pieces together. You will need to write a simple planner to control the motion of the arm and implement states in the state machine to control the arm for picking and placing blocks. Given a set of target locations from the block detector, your arm will need to servo into a position from which it may grab the target block, then servo to the drop-off location. The actual end effector position may be different than the target locations given by the block detector, so you should take this into account.

This will be similar to the teach and repeat planner, except you will need to automatically generate the waypoints based on the output of the block detector, kinematics, and your strategy for the final competition.

|  |
| --- |
| **6.1**: Implement click to grab/click to place where you can click on a block in the video and the arm will move to grasping location, then a second click will tell the arm to move to a drop off location. Initially you only need to implement this for blocks on the ground plane, because it will be useful for tuning performance in the competition. CHECKPOINT:   * Demonstrate at least this minimal click to grab/click to place to an instructor. If you also integrate the gripper at the same time, show them that instead.   REQUIRED FOR REPORT:   * Describe the algorithm you used to create and execute the Rexarm motion plan. |

|  |
| --- |
| **6.2:** Integrate your gripper into the planner, so you may execute grasp and release functionality. If your gripper will need to grasp differently depending on the location on the board of the target object then you should include this logic in the planner. REQUIRED FOR REPORT:   * Describe the process of integrating the gripper system to the Rexarm and how you implemented the gripping logic. |

|  |
| --- |
| **6.3**: Implement logic for the final competition into your state machine.  REQUIRED FOR REPORT:   * Evaluate the performance of your system during the competition * Discuss what improvements could be made to improve performance |

# Competition - Blocks Pentathalon

|  |
| --- |
| **Event 1: Pick n’ stack! (100 points)**   * 3 Blocks (Red, Green, and Blue) will be placed on the right half of the board (not stacked) with >2.5cm of space in between * Move the 3 blocks to the other side of the board and stack them at any location * 15 points for each block picked * 15 points for each block stacked * 2.5 points for each 10s less than 40s it takes to complete the task (up to 10 points) |

|  |
| --- |
| **Event 2: Wacky Pick n’ place! (150 points)**   * A block will be placed in a special fixture at a specific orientation * Move the block from one fixture to the other and place at a different orientation * Repeat 2 more times in different locations * 25 points for each block picked * 25 points for each block placed |

|  |
| --- |
| **Event 3: Line ‘em up! (150 points)**   * Blocks will be placed in a specific configuration, some will be stacked (no more than 3 high) * Order the Blocks in a line in the color order:   + Black Red Orange Yellow Green Blue Violet Pink * Blocks must be within 1 cm of one another and within 1 cm of a straight line * 15 points for each block in the correct order (5 if block is out of place) * Up to 30 points for the neatness of the line (judged by instructors) |

|  |
| --- |
| **Event 4: Stack ‘em high! (200 points)**   * Blocks will be placed in a specific configuration, some will be stacked (no more than 3 high) * Order the blocks in a stack in the color order:   + Black Red Orange Yellow Green Blue Violet Pink * 20 points for each block in the correct order (5 if block is out of place) * 10 points for each 20s less than 120s it takes to complete the stack (up to 40 points) |

|  |
| --- |
| **Event 5: Block Mover (200 points)**   * Blocks will be placed at approximate location of ( -200mm, 200mm) * Grab the base block and slide it around 3 points of a square * ( -200mm, 200mm) -> ( 200mm, 200mm) -> ( -200mm, 200mm) -> ( -200mm, -200mm) * Do not change the orientation of the block as you do this * Record the path of the block using your kinect for including in the report * Repeat the challenge with a stack of blocks (3+) without it toppling over * How tall a stack can you do? |

Your score will be the sum of your best run on each event. The top 4 teams will receive extra credit on the assignment [+8, +6, +4, +2] points.

# Report Rubric

A compelling argument needs to be provided on the capabilities of your group’s robot control. The course staff will evaluate the quality of a group’s work based on four major criteria from in-lab demonstrations, the project competition, and the submitted report. These criteria and corresponding metric-oriented questions are outlined below:

* Gripper Design
  + Describe the design and design choices/tradeoffs you made.
  + Discuss any issues that became evident in the design and how you revised it.
  + Include drawings or a 3D rendering of your gripper assembly.
  + Submit an archive containing design files and STL files.
* Teach & Repeat / State Machine
  + Describe how you implemented the path smoothing
  + Generate a plot of the robot end-effector position in world coordinates at slow and fast speed with and without path smoothing (4 curves)
  + Generate a plot of the magnitude of end-effector velocity with and without path-smoothing at high speed
* Kinematics
  + Include a DH table for the Rexarm in your report
  + Describe how you verified that your arm tooltip achieved expected workspace coordinates for your test cases, estimate the error, and justify the accuracy of your measurements.
  + Include a schematic diagram of the arm with all relevant FK and IK parameters
  + Precisely specify the IK function by writing the equations, explaining them, and referencing the source file and lines
* Camera Calibration & Block Detection
  + Report your intrinsic matrix and depth calibration functions
  + Describe how you verified the calibration was correct and provide evidence for the accuracy of your calibration
  + Describe the methods you used to implement the block detection and any strategies you implemented to limit false positive detections and enhance the robustness of the detector
  + Describe how you verified the accuracy of your block detector and provide evidence of its performance. Create a plot that visualizes the uncertainty vs. location on the board.
* Final Bits & Pieces
  + Describe the algorithm you used to create and execute the arm motion plan.
  + Evaluate the gripper performance (whether the default or a custom gripper) and mention any possible improvements .
  + Evaluate the performance of your system during the competition
  + Discuss what improvements could be made to improve performance

# 

# Checklist

Demonstrating basic functionality will be 25% of your grade. You will be given a checklist for instructors to sign off on. Keep this over the course of the project and scan it when completed. In order to ensure you are completing the lab on schedule, demonstrate the following to the instructors (in no particular order) **before 5pm on Friday 10/4:**

* Show your gripper to the instructors and discuss your design with them
* Demonstrate to an instructor your teach-and-repeat functionality
* Demonstrate to an instructor your calibration procedure
* Demonstrate at least a minimal click-to-grab/click-to-place to an instructor. If you also integrate the gripper at the same time, show them that instead.

# Report Template

We have created an Overleaf project to be used as a template for your reports. The read-only project can be found [here](https://www.overleaf.com/read/rgtzswgvmbgg), and you should clone and use it as a template for your group’s project. Please preserve the paper organization given.

# Submission

The files to be submitted:

1. PDF document of report w/ appendix
2. source code (run make clean first)
3. design files and stl files for your gripper
4. a scan or photo of your signed checklist
5. a scan or photo of your honor code certification (see below)

Please submit a .zip or tar.gz file to Canvas with the following structure. Only one student should submit the archive.

# = team number as 2 digits

* <#>-armlab-f19.zip
* <#>-armlab-f19/
  + report/
    - <#>-armlab-f19.pdf
  + code/
    - armlab-f19/
  + gripper/
    - <gripper solidworks/inventor files>
    - <gripper stl files>
  + requirements.csv
  + checklist.png/pdf
  + signature.png/pdf

# Certification and Peer Evaluation

[required for credit, 0 points]

Print or write the following on a sheet of paper:

**"I participated and contributed to team discussions on each problem, and I attest to the integrity of each solution. Our team met as a group on [DATE(S)]."**

Each team member should sign the paper (physically, not digitally), and a scan/photo of the paper should be included with your report. By doing so, you are asserting that you were materially involved in each answer, and that you certify that every solution complies with the collaboration policy of this course. In the event of a violation of the course's guidelines, your certification will be forwarded to the honor council.

If the statement requires qualification (i.e., it's not quite true), please describe the issue and the steps you took to mitigate it. If a team member did not materially contribute to a particular solution, then their score on that problem will generally be reduced. The staff will use their discretion based on the specific circumstances you identify, and we are happy to discuss potential problems prior to the problem set due date.

If your team works together, as required, this will be painless! :)

Each member of your team must also complete a peer evaluation, which will be posted after the report is submitted. Your evaluations of your teammates will remain private. If problems develop with your team, it is up to your team to resolve them: we will not intervene except in the most extreme situations.