Project ROBSIM

ENSC-488 Introduction to Robotics

In this project, you will simulate a 4-DOF SCARA type (RRPR) manipulator and apply the subject matter of your lectures to the manipulator. The SCARA Configuration, shown in Figure 8.5 of your text, is the real robot is shown in Figure 1, and a schematic of the real robot is shown in Figure 2. This simple manipulator is complex enough to demonstrate the main principles of the course while not bogging you down with too much complexity. Each part of the project builds upon the previous one so that at the end of the project, an entire library of manipulator software to drive the robot will be created. Two SCARA robots are setup in the lab, and teams will test their custom programmed functions on the real robot (if their functions safely pass all tests in simulation). The project is to be done in groups, with each group consisting of **three students**.

Teams are provided an emulator. The hardware (actual robot) as well as the emulator has the same simple programming interface. The emulator provides a graphic display for your robot; hence, allows you to work on the project without worrying about hardware. The very same code that works for emulator will work on the real robot. You may use the hardware emulator to test your modules before integrating all modules. A set of header and library files needed for your project have is posted on CANVAS.

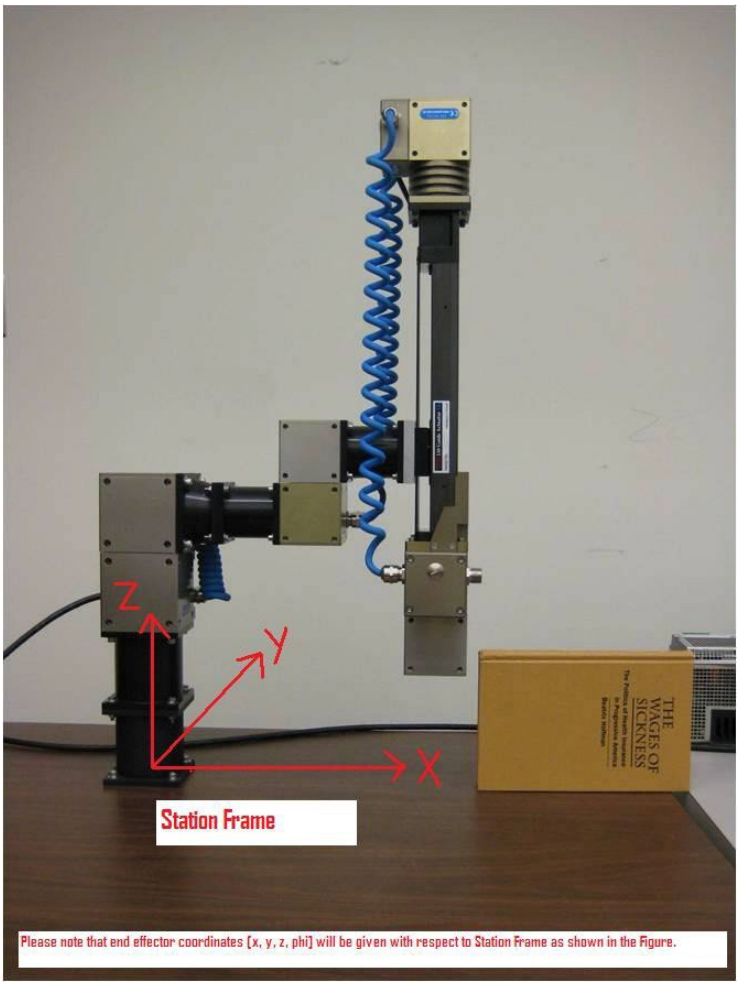

Figure 1: Assigned Station Frame. The robot is shown in joint configuration (0°, 0°, -200mm, 0°).
Note: The pose/orientation of the station frame.

1. Component modules

The component modules of ROBSIM are:

1. **Basic Matrix Computation Routines:**
   - Build the basic routines as described in programming exercises (part 2: 2,3,4) in your text. Please note that the position vector (in your case) will be a 3-tuple (*x; y; z*) and the rotation matrices will be 3 x 3, with the additional restriction that all rotations are around *Z-axis*. You may use public domain source for matrix routines where available.
     DO NOT *include external libraries/headers* as these libraries are not installed in the robot lab, where your code will be compiled prior to moving the manipulators.

2. **Forward and Inverse Kinematics**:
   - Assign D-H frames to the manipulator. Assign station frame {**S**} as shown in Figure 1. Assign the origin of Tool frame {**T**} in the center of the gripper (note $\boldsymbol{Z}_T$ points downwards). Determine the D-H parameters and the forward kinematic equations. Although numerical values are given, derive forward and inverse kinematics with symbolic values and then substitute the numerical values as appropriate parameters in your code.
   - Implement the procedures *KIN()* and *WHERE()* as in programming exercise (part 3: 1,2).
   - Implement the procedures *INVKIN()* and *SOLVE()* as in programming exercise (part 4: 1,2).
   - Note: Routines programmed must respect joint limits, and in case of multiple solutions, choose the most appropriate solution (closest to the current configuration). Thoroughly test routines and make sure that they are bug-free.
     - Use the **graphic display** - using *DisplayCon_guration()* command - as a debugging aid!

3. **Trajectory Planner or Collision-free Path Planner:**
   - Implement a joint space trajectory planner (similar to Part 7:1,2; but teams choose an appropriate type of spline) capable of generating trajectories corresponding to the following command:
   - *MOVE_ARM* to {**G**} via {**A**}, {**B**}, and {**C**} with total duration = T seconds; where {**A**}, {**B**}, {**C**} and {**G**} are TOOL frames specified with respect to. STATION frame {**S**}.
   - Selection can either apply parabolic blends or cubic splines. The **solution must have velocity continuity** in the trajectory planner (i.e. acceleration can be discontinuous; a nicer solution is if acceleration is continuous but this is not required). As a debugging aid, plot the *x* and *y* components of the trajectory on an *x-y plane* with time as an implicit parameter. This "overhead view" will help teams visualize the trajectory better since the emulator display does not have the top view of the trajectory.
   - Joint space trajectory planning is the minimum. But if teams implement Cartesian space trajectory planning (instead of joint space), that merits a 5% bonus.
     Note that it requires Jacobian implementation.

4. Dynamic Simulator:

- Implement a dynamic simulator for the manipulator as in programming exercise (part 6:2). Teams should use Maple/Mathematica/MATLAB for this and must use a "convert" function in the applicable software (all applications support it, although the precise name for the function may be different) to directly get c-code for the symbolic expressions. Otherwise, there is a good chance of introducing bugs if you input the expressions manually in your code. Again, as in kinematics, deriving the dynamics equations using symbolic values (at least for mass and length parameters) and then substitute the numerical values. Thoroughly test the simulator by itself since this will act as your simulated robot. One example is to apply a torque to Joint 1 only and see the resulting motion on the emulator.

5. Controller:

- Implement a trajectory-following control system for the manipulator. As a guide- line (you can choose other values), set the servo gains to achieve closed loop stiffness **(k_p)** of **175.0**, **110.0**, **40.0** and **20.0** for ***joints 1 through 4*** *respectively*. Try to achieve approximate critical damping. Put appropriate limits on the maximum torques and forces that joint motors can apply.

6. User interface:

- The user interface part of your system should have the following capabilities.
- **Note:** the UI is NOT the project's focus so do not waste much time developing a fancy UI. For example, Teams can simply read data from a file or type a character to select a function. Functionality counts for the UI.
  a) The system should be able to input and output the Tool frame {**T**} specified in terms of [*x; y; z;* ϕ] with respect to Station frame {**S**}.
  b) The user should be able to specify the initial robot configuration (either by specifying the joint angles or by specifying the tool frame) and move the robot to that configuration.
  c) The system should be able to graphically display the arm motion and workspace obstacles.
  d) The system should be able to display plots of joint space trajectories and torques. You would use MATLAB/Excel/GNUplot for this purpose. The joint and torque limits should also be displayed on the same plot so it is visually easy to see if limits are being exceeded.
  e) The system should be able to detect when joint limits or torque limits are exceeded. The system should inform the user should such an event happen.

Integrate the above modules into a robot programming system. See Figure 3 for a block diagram of the overall system.

2. Project Phases/Demos

The project will be demonstrated in incremental phases.

1. **Phase/Demo 1**. Implement and demonstrate Forward and Inverse Kinematics (KIN/WHERE and INVKIN/SOLVE functions) on both the robot hardware and emulator. User should be able to specify the relevant frames or joint vectors, and a team's system should output the computed frames or joint vectors, as the case is.
   a. In case of WHERE, the system, given a desired joint vector, should output the corresponding position and orientation of the tool frame with respect to the station frame in terms of a four tuple (*x; y; z;* ϕ) and also show it graphically.
   b. In case of SOLVE, the system should, given a desired (*x; y; z;* ϕ), determine the joint configuration **closest to the current joint configuration** and move the robot to that configuration (the tool will be in the desired (*x; y; z;* ϕ) after the move).

   At the end of this phase, teams should be able to demonstrate a *pick and place task*.
   a. A simple solid object (i.e. a wooden cube, for example) would be the object to be grasped. From the given start gripper frame (say, {$\mathbf{T}_0$}). First, command/move the robot to a given gripper pose, say {$\mathbf{T}_1$}, that is directly over the object to be grasped. Second, move the robot down (slowly!) so that the object to be grasped is directly between the gripper jaws (called {$\mathbf{T}_2$}) and then close the gripper.
   b. The robot will be commanded to move up to {$\mathbf{T}_1$}, and then go to a location (say {$\mathbf{T}_3$}) that is directly above where the object is to be placed. The arm will then move down (again slowly!) to {$\mathbf{T}_4$} and open the gripper to release the object from the grasp. The robot will then move back to {$\mathbf{T}_3$} and then back to its initial position {$\mathbf{T}_0$}.

   Demo 1 date: Check Canvas. Teams will need to sign up for a demo slot on CANVAS.

2. **Phase/Demo 2**. Trajectory/Path Planner: Implement a trajectory/path planner to demonstrate on both the robot hardware and the emulator.
   a. For the trajectory planner, user specifies three intermediate Tool frames (w.r.t the station frame {**S**}), i.e. ${}^{S}_{T_1}\boldsymbol{T}$, ${}^{S}_{T_2}\boldsymbol{T}$, ${}^{S}_{T_3}\boldsymbol{T}$, from an initial frame, ${}^{S}_{T_0}\boldsymbol{T}$, and a goal frame, ${}^{S}_{T_g}\boldsymbol{T}$
      - Code should allow a user to start at an initial pose, pass through three intermediate tool frames, and end at the goal frame.
   b. A team's system should plan and execute a trajectory that starts from rest at the current arm configuration, passes through (or close to) the intermediate frames, and stops at the goal frame.
      i. Teams can demonstrate the same pick and place task as in Demo 1, but this time the gripper frames $\{\boldsymbol{T}_i\}$'s will act as *points / waypoints*, and the trajectory will be determined by your trajectory planner.
   c. If a team implements a collision-free path planner with obstacles, this planner should be demonstrated on both the robot hardware and emulator. User specifies the TOOL frame ${}^{S}_{T}\boldsymbol{T}$ at start and at goal. User also specifies the obstacles in the environment. The system should plan and execute a collision-free trajectory that starts from rest at the start frame and stops at the goal frame.

   Demo 2 date: Check Canvas.Teams will need to sign up for a demo slot on CANVAS.

3. **Phase 3a**. Dynamic simulator, implemented on emulator only.
   Teams need to show the behaviour of your robot dynamic simulator under predefined (e.g. constant) torques/forces.

4. **Phase 3b.** Fully-functioning ROBSIM system with controller, implemented on emulator only.
   Besides the above modules/functions, teams need to show the behaviour of their selected controller. When a trajectory is executed, record the error and torque history for each joint for the entire motion; and show the plots (after the motion).

5. **Final Project Demo 3**: Integrates Phases 1-3 and demonstrates it on emulator only.
   a. For the trajectory planner, the user specifies the start frame, the goal frame, and three intermediate frames (w.r.t the station frame {**g**}),${}^{S}_{T}\boldsymbol{T}$. A team's system should plan and execute a trajectory that starts from rest at the start frame, passes through (or close to) the intermediate frames, and stops at the goal frame.
   b. As the trajectory is executed, record the error and torque history for each joint for the entire motion; show the plots (after the motion).

   Demo: Date. Check Canvas. Teams will need to sign up for a demo slot on CANVAS. A project report (one per group) about 15 pages of text plus figures and plots) should be submitted at the time of project demonstration. See Section 4 for details.

3. Details

Robot parameters

1. For the project, the manipulator is mounted so that all joint axes are vertical.
   Assume that $\hat{Z}_0$, $\hat{Z}_1$, and $\hat{Z}_3$ point vertically upwards, and $\hat{Z}_3$ and $\hat{Z}_4$ point vertically downwards.
2. The ***kinematic parameters*** are shown in Figure 2.
   i. **Joint 1** has a joint limit of ±150°,
   ii. **Joint 2** has a joint limit of ±100°,
   iii. **Joint 3** can move between [-200mm, -100mm] (the lower limit is hardware limit, but the upper limit is artificially imposed so that the gripper does not hit the workbench),
   iv. **Joint 4** has a joint limit of ±160°. In your simulator, assume that the robot has a gripper mounted at the end.

3. The ***dynamic parameters*** are as follows.
   Note that this applies to the simulator only, and not the real robot.
   i. m_1, m_2, m_3, m_4 are masses of the four respective links.
   ii. m_1 **= 1.7kg,** m_2 **= 1.0kg,** m_3 **= 1.7kg,** and m_4 **= 1.0kg**. For all four links, assume that the mass is all concentrated into a point mass, and the respective locations of centers of mass are shown in Figure 2.
   iii. Assume some viscous friction at each joint. Of course, gravity acts vertically downward. A good idea suggestion is to derive all equations using a parameter *g* and make *g* user defined. This way you could test your robot's behaviour in space (zero gravity)!

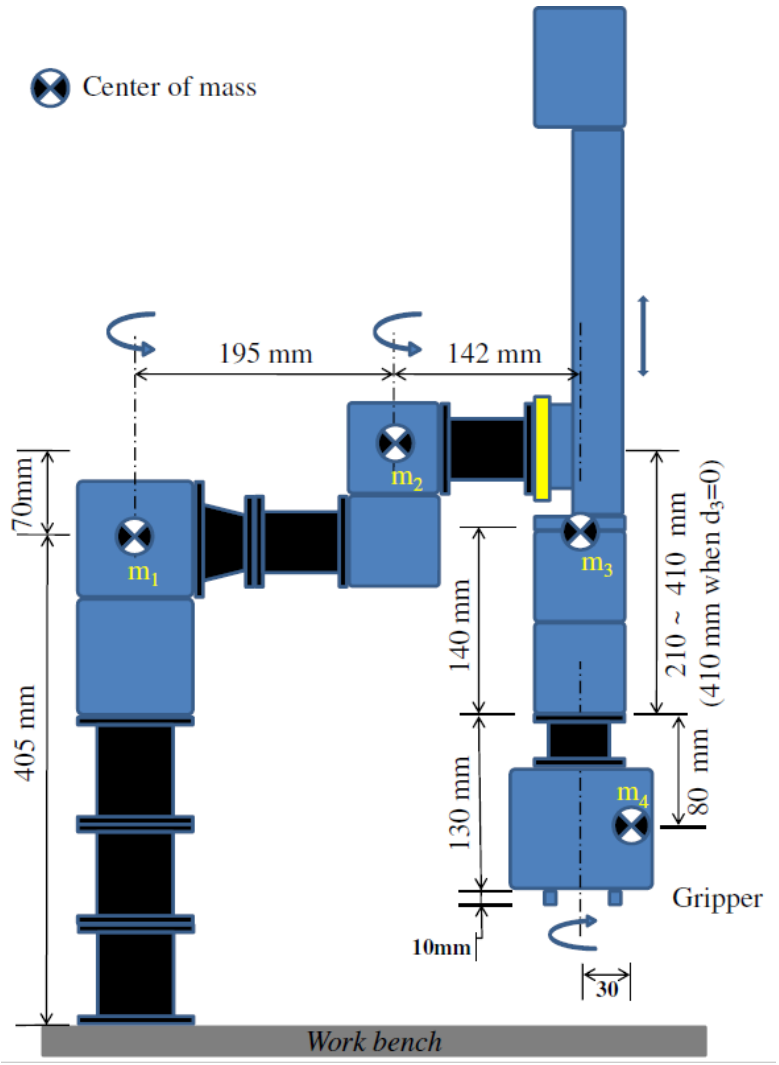

Figure 2: Schematic of SCARA type robot system assembled in 488 Lab.
Note: The instructor may change the COM for link 4, i.e., location of m4, to further simplify dynamics calculations. They will let teams know prior to Demo 3 when needed.

Hardware/emulator APIs. As shown in Figure 3, teams will be using different functions to "drive" the hardware/emulator in different demos (i.e., (A), (B) and (C)). Do make sure to use appropriate move/display command.

1. ***MoveToConfiguration(conf)*** for (A), to move the robot to a given configuration. This function will use maximum velocity and acceleration for motion and used for both hardware and the emulator.
   -***Teams must use this function for driving the robot in Demo 1***.
2. ***MoveWithConfVelAcc(conf, vel, acc)*** for (B), to move the robot with desired velocity (vel) and acceleration (acc). This function is for both hardware and emulator.
   -***Teams must use this function for driving the robot in Demo 2***.
   - *Arguments:* double arrays of Size=4, corresponding to 4 joints

   Note: Due to the limitation of hardware programming interface provided by the manufacturer, only the velocity (vel) argument is used. Because of this simplification, there may be a slight drift at the end of the trajectory.
3. ***DisplayConfiguration(conf)*** for (C), simply display the robot under the given configuration. This function is for emulator only.
   -***You must use this function for driving the robot in Demo 3***.
4. ***Grasp(close)***, to close (with *close=true*) or open (with *close=false*) the gripper.
   This function is for both hardware and emulator.
5. Other functions are provided for general purpose, and they are all for both hardware and emulator.
   - ***GetConfiguration(conf)*** to retrieve current robot configuration.
   - ***StopRobot()*** to stop the robot.
   - ***ResetRobot()*** to reset the robot after calling *StopRobot()*.
     - This enables the robot to respond to move commands again.

Note: Teams must use the command corresponding to the letter at the output of the module as shown in Figure 3 to move the robot—real robot or the virtual robot (emulator).

In above functions, the arguments used for configuration/velocity/acceleration are double arrays of size 4, corresponding to 4 joints. Please read the ***provided header file ensc-488.h*** for more details.

Review the unites used for arguments:

*For **revolute joints***, they accept:
- *degree* for joint value,
- *degree/s* for velocity,
- *degree/s*^{2} for acceleration;

For **prismatic joints**, they accept:
- *mm* for joint value,
- *mm/s* for velocity,
- *mm/s*^{2} for acceleration

WARNING!!!!: Some programming and implementation advice
(for those who may not have much programming experience)

The project involves non-trivial amount of coding. Therefore, follow **structured programming approach**. Adopt a **modular structure** for your code with each functional unit represented by a separate module. **Test** each module thoroughly before integrating it with the rest of the system. **Parameters such as link lengths, joint limits, default torque maximums, etc. should reside in one place and should be easily accessible and modifiable**. It may also make sense to use global variables for certain parameters. Believe me, following a disciplined approach to coding will save you considerable headache and many sleepless nights in debugging!

4 Project Report Submission

A project report (one per group) not exceeding 15 pages of text (plots are extra) is due at the time of project demonstration. The report should adhere to the following guidelines:

- **A succinct description of each module**. There is no point repeating material from text.
  Teams must include all FINAL results of each module (for example: *frame attachment, D-H parameters, expressions for inverse kinematics, dynamic equations, control laws, and relevant block diagrams with sampling times*).
- **Include plots** of trajectories, velocities, accelerations, torques for a sample trajectory. Make sure the plots are correctly labelled.
- Any **anomalous behaviour** that you observed in your simulations/experiments and an explanation for it if you have one.
- The report should **clearly state the role of each member** of the team and the work completed by the member (about a paragraph for each member).
  - Also state the methods used by the team for communication between members for various aspects of the project—for example for code development, code sharing, etc. Each member must have a non-trivial contribution code development.
- **Please review the course policy for plagiarism.** *ANY* plagiarism (i.e. copying or patch writing) in the report (or code) will result in a zero for the *entire team* (i.e. the group claiming the submission is original work).

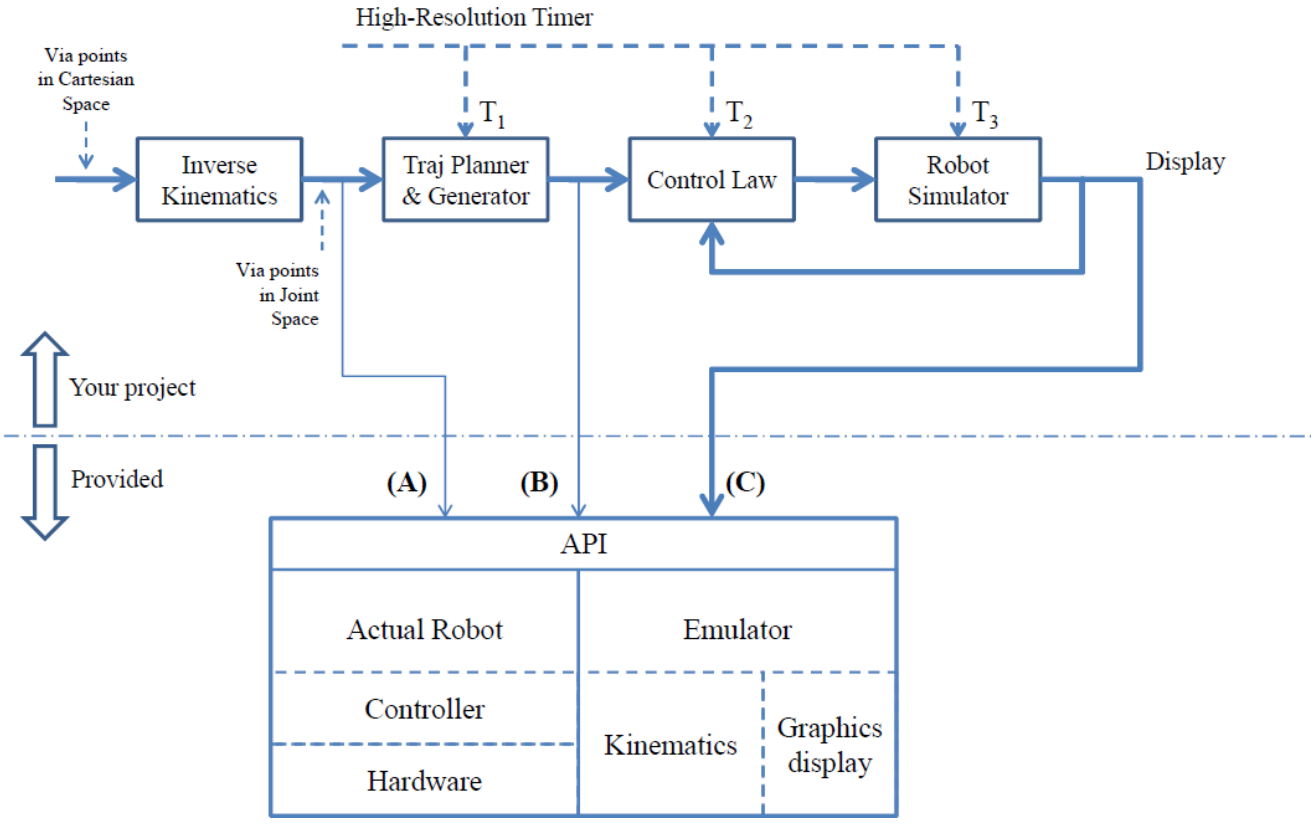

Figure 3: Block diagram of your robot programming system: ROBSIM

Appendix A: Printable Figures of ENSC 488 Robot

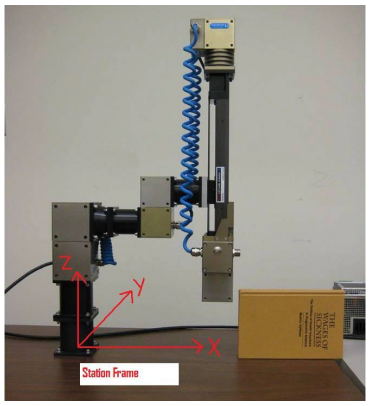

Figure 1: Assigned Station Frame.
The robot is shown in joint configuration:
(0°, 0°, -200mm, 0°).

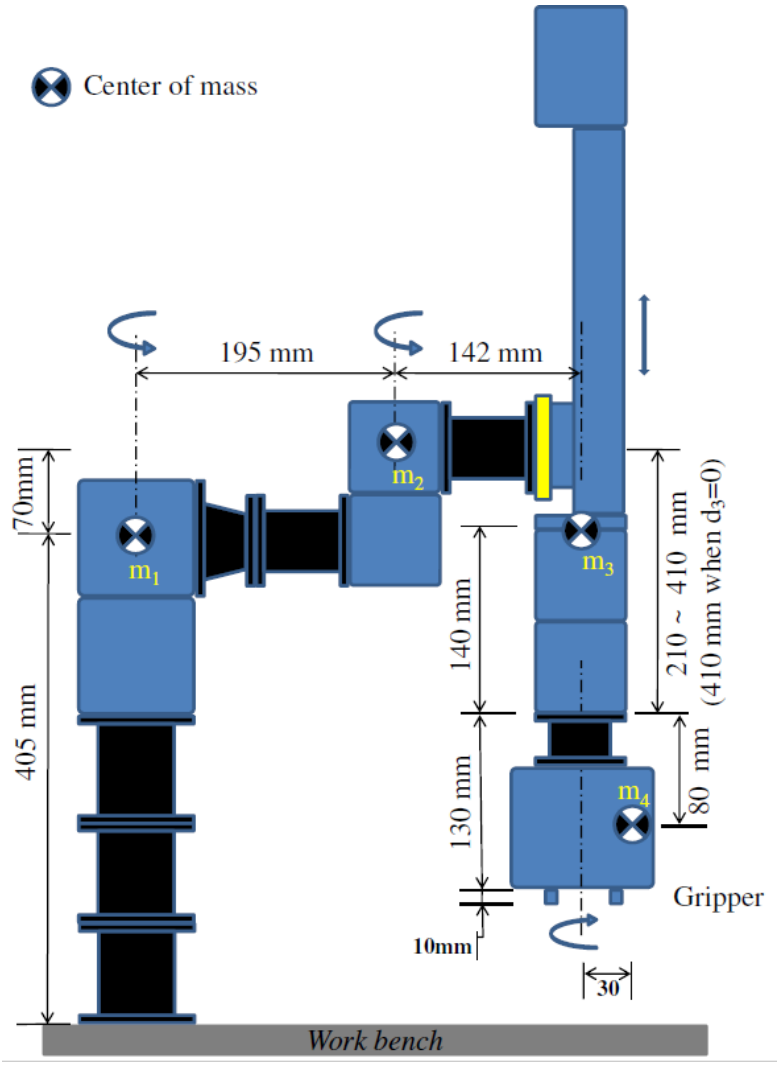

Figure 2: Schematic of SCARA type Manipulator