[Company name]  [Company address]

Library Bot documentation

Contents

[To do 2](#_Toc101351073)

[Code structure 3](#_Toc101351074)

# To do

**03/05/2022**

* For collision detection, add a way to detect a *moving* object (i.e. the book rectangular prism), to avoid collision with the environment. In each frame, get the global transform of the book (visual servo opportunity?), and check its collision with any other objects (or EF arm). This could be done by passing the transform of the book each step to the ‘obsticleConstructor’, ~~making the vertices and faces, then checking those for collision.~~ Finding the 12 lines which comprise the shape (simply the vertexes of each shape, 12 lines for object edges, then an addition 6 for the faces to be triangles). Then collision detecting these with the shapes in the environment at EACH STEP in the robot movement.
* Add collision avoidance (check week 5).
* Create a class for each object (obstacle and books) which contains its global transform. This class will also plot the object into view (good for static objects like the table, shelves, stationary books). Also, for the motion of the book, we need a function which can animate the book moving along a path (the path is simply the transform of the book at each qMatrix frame).
* Plan out the path (a literal 3D line) which we \_\_\_\_ the robot to follow between set points (this will be achieved in practice using the RMRC A to B function). Then, for area’s where we must calculate the robot’s position (e.g. depending where the book goes), plan out a 3D path which is a function of the desired location. This could, for example, make the robot move from a set point to a calculated transform (and back) in order to place the book.
* Create the 3D models for the robot (HansCute), and figure out the PlotAndColour() function in the robot constructor.
* Within the RMRC function, figure out *the best* way to plot a qMatrix of the robot (not what I (Patrick) did for A1, where I recursively called plot3d over and over).
* Add 3D modelling for signage and barriers. Design the workspace setup [Make it look like a library(ish) wish other shelves, coloured ground, etc. Make the robot sit on a table, which is fenced off. Have the e-stop nearby. Use a method to detect if a person (arm, head, etc) enters the robots workspace, and trigger the ‘collision detection’ emergency stop procedure.
* Include a Matlab graphical user interface (GUI) to interact with the system. The GUI should have advanced “teach” functionality that allows jogging the robot. It should include both individual joint.
* movements (like the Toolbox’s “teach”) plus enable [x,y,z] Cartesian movements.
* Watch an example video of what is due in week 11. Is it the complete system? Is it simply a well put together video of the robot in action?
* Organise a day to come into uni and record the real robot, moving the books. Therefore beforehand we need to make physical (cardboard or something similar) books to be moved by the robot. Also, we should make shelves? Discuss with tutors.
* Once some more of the above is implemented, we should add visual servo-ing for the placement of the book within the shelf (i.e. the book has to move perfectly in a straight line, into the slot).

# Completed To-Do items

* Get DH parameters for robot and make class in Matlab (Nhan)
* Setting up environment (Nick)
* Setting up functions and classes (Patrick)

# Code structure (needs updating)

Hans cute class:

|  |  |  |
| --- | --- | --- |
| Create robot (constructor with base location) | **Done** | HansCute(baseLocation) where baseLocation is a transformation matrix (can be left blank, to default to 0,0,0). |
| Set base location (alternative to above) | **------** | Already a model.base() function by default |
| Get current end effector position (fkine) | ------ | model.fkine(q) |
| Grab function (for end effector) | ------ | Self.gripperBool = true/false |
| Get grab status | ------ | Self.griperBool (returns true or false) |
| Get position of EVERY joint (useful for collision detection) |  |  |
| Set joint limits | ------ | Done within class constructor, in DH params |
| Collision detects with surface (input surface [by vertexes?]) (iterative to check every line of each robot arm, with each surface) |  |  |
| Stop robot movement |  | Function which stops robot movement. Used for both collision and estop. Therefore, one variable is for the stop function, and another indicates whether it was a collision or estop. |
| Move from point A to B in a linear fashion. MUST obey some predefined joint limits (defined within the function), and constantly be checking for collisions. Use RMRC |  |  |
| Start operation of robot. The flags for collision must be off for this |  |  |

Graphical user interface, text, application, letter

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**Collision detection process DONE**

We will take a proposed q matrix (7\*n), and check if ANY of the proposed states collide with any of the triangles (objcts).

1. Have an object to represent a rectangular prism as an object. Can be constructed either with upper and lower (opposite) corners, or a centre with x,y, and z lengths. Has a function to return vertexes, faces, and face normal.
2. Also have a means to get the transform of each link of the robot (7DOF, therefore 8 links).
3. Use lineplaneintersection.m code to check for collisions.