EN 601.656 Computer Integrated Surgery II

Final Report: Automatic Calibration of Mosquito Dissection System for the Production of Malaria Vaccines

05/01/2022

Miles Liu

Mentors: Balazs Vagvolgyi, Anna Goodridge

Technical Summary

Project Background, Importance, and Goals

Sanaria Inc. and the team at Computer Integrated Interventional Systems Laboratory (CIIS) have been working together to develop a robot platform that can fully automate a critical step in malaria vaccine production - salivary gland extraction. Once deployed, this robot platform can significantly increase gland production rates, and thus eliminate this vaccine production bottleneck.

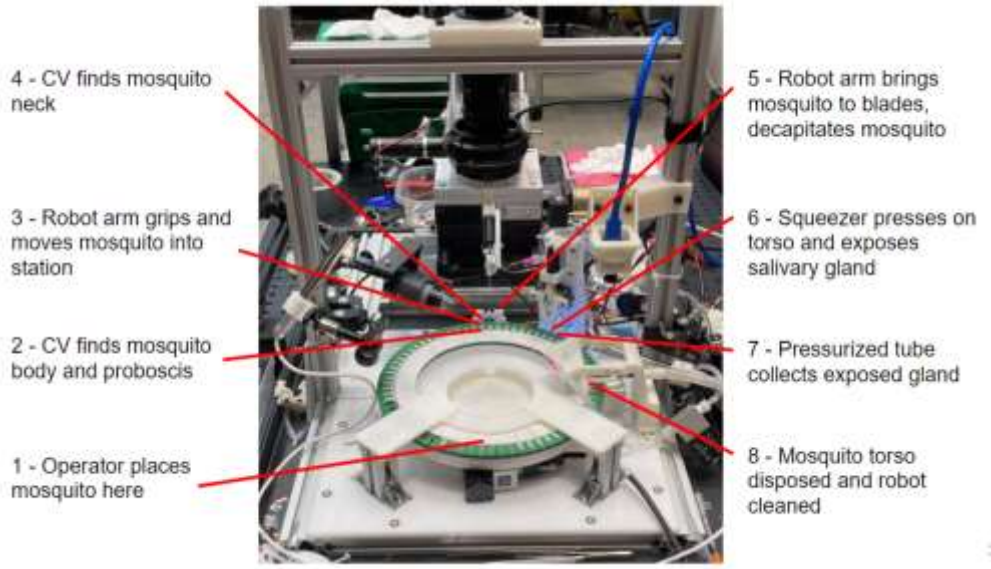

Figure 1: Current robot design, with steps of mosquito manipulation process

As the robot requires high precision mosquito manipulation movements, it will also require precise calibration for many of its components. As the robot is continuously developed, every time hardware parameters are modified, the operator would need to perform a full recalibration of the robot. While there are existing procedures for each of these calibration components, they are fully manual processes, which can be time and labor intensive.

The goal of this project is to streamline calibration routines and create a well-integrated single calibration package that can handle most calibration tasks automatically. This project also aims to rethink how the robot workspace and its planned paths are defined, and implement a solution that can be maintained easier by human developers.

Specifically, there are three calibration tasks that lie within the scope of this project:

- Robot homing
- Robot waypoint calibration (path planning with respect to workspace)
- Handeye calibration (camera to robot arm)

Each of these calibration workflows require their own callable procedure, as well as a graphical user interface to facilitate interaction from the supervising operator. These components will serve as the main deliverables for the project.

Technical Approach & Results – Robot Homing

The goal for the **robot homing calibration** task is to obtain a mapping from the robot's encoder frame to a fixed frame in the robot's workspace, or the "Home position". To achieve this, there needs to be a procedure that can move the robot to the fixed "Home position" in the robot workspace without using the robot's encoder positions.

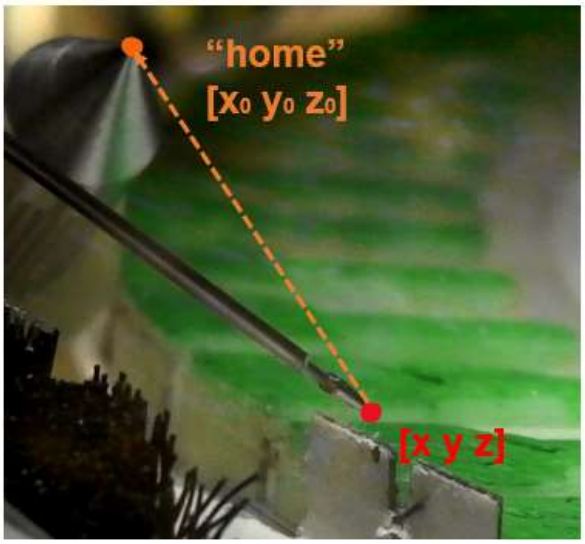

Figure 2: Robot homing in one camera view

To achieve this, we opted to develop a visual servoing algorithm for moving the robot to any arbitrary position in the cartesian robot workspace. The visual servoing procedure uses two cameras (one overhead and one at an arbitrary position) to capture two 2D projections of the robot's tooltip and goal positions in images. The pixel positions (u, v) of the goal and current gripper tooltip can then be extracted from the images using an automated computer vision component or a manual procedure. With this image processing infrastructure, the algorithm can then iteratively loop through motion steps to move the gripper tooltip to the target position, until the gripper tooltip and the target on both images converge, meaning that the tooltip is exactly on top of the goal position. At each of these motion steps, the robot performs a small motion in each of the axis (X, Y, and Z), finds the new tool position in the images, and determines the gradient of the tooltip's pixel locations with respect to the small motion in each of the robot's axis in the form of a Jacobian. This Jacobian can then be fed into a minimization algorithm (Powell gradient descent in our implementation) to obtain a new position in robot coordinates that would minimize the distance between the goal position's image projection pixel coordinate and the new robot position's image projection pixel coordinate. Since there are non-negligible radial and other distortions in the images being captured, the Jacobian we obtain at each step is only an approximation for the local Jacobian. Therefore, the algorithm uses multiple steps as well as a max distance that the robot can traverse in each step.

The algorithm itself is implemented as a ROS node, and needs to be operated directly through an operator GUI. The GUI provides the operator with the ability to start, stop, and monitor the homing procedure.

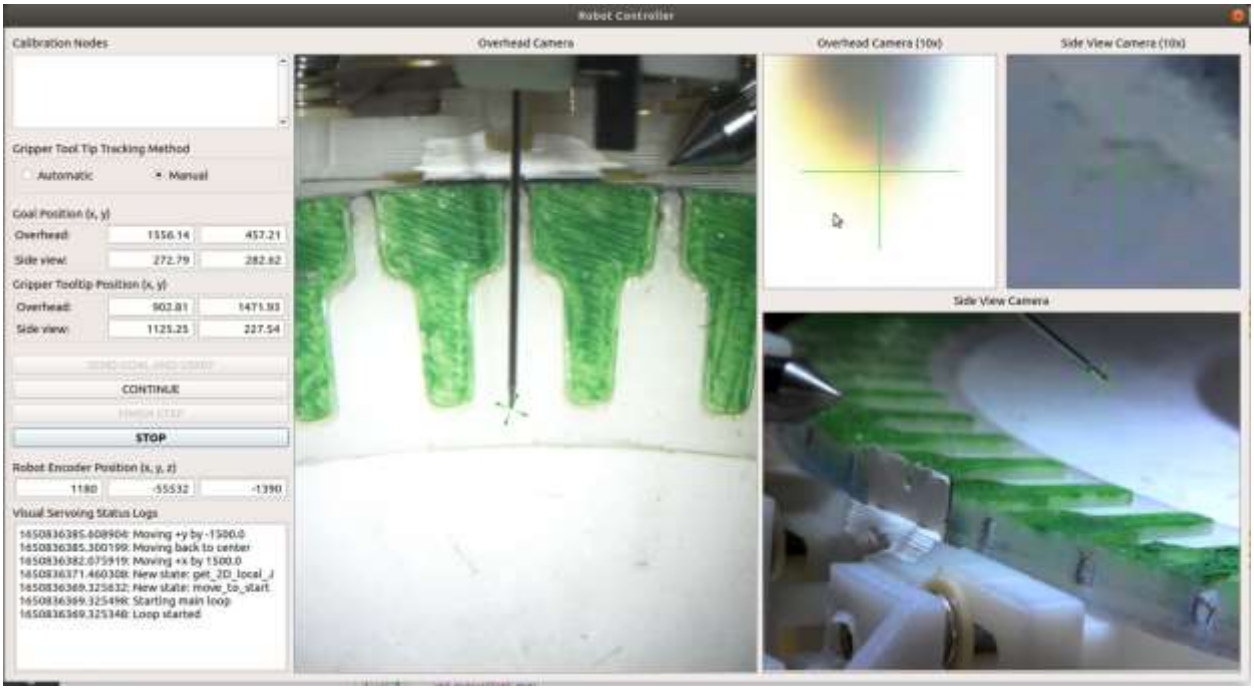

Figure 3: Robot homing GUI

One subcomponent of this robot homing algorithm is extraction of gripper tooltip positions from images. The ideal method is by using a dedicated computer vision service that can automatically return the pixel coordinate of the tooltip within a certain tolerance when given an image. However, we also developed another method that allows the operator to label the gripper tooltip on a given image on the GUI. This manual procedure is more time consuming and labor intensive but allows the visual servoing algorithm to run when the computer vision service is not available or not accurate enough.

The current implementation of the robot homing procedure does not have the CV gripper detection component integrated in, so robot homing is currently done with manual labeling. A full procedure takes approximately 2 minutes to complete, and requires approximately 38 operator interactions total (clicks, mouse drags, etc.). It can bring the gripper consistently within ~200 encoder counts (100 microns) of the target position.

Future Work – Robot Homing

The two main items that can be improved is the autonomous capacity of the procedure and the accuracy. Once the automatic gripper detection computer vision component can consistently and correctly label the gripper positions in images, the operator may only need to send a command to start the procedure, and the algorithm will take care of the rest. This can also drastically increase the speed of the procedure, since the majority of the currently procedure's time is spent waiting for the operator to label the gripper positions. The accuracy of the procedure is directly tied into

the motion fidelity of the robot's motions, which is being explored in a separate project. With improved robot motion accuracy, the accuracy of this procedure is expected to also increase.

Outside of just robot homing, the visual servoing algorithm can also be used for other tasks that may require the robot to move to a specific location identified in two cameras without knowledge of the encoder positions. A possible application can be other calibration tasks, such as that explored in the waypoint calibration component of this project.

Technical Approach & Results – Robot Waypoint Calibration

The goal for the **robot tool path calibration** task is to develop a more intuitive, human understandable, and unified data structure for the representation of waypoints used in the robot's normal operation.

In order to achieve this, we developed a custom data structure that does not explicitly define each robot waypoint as its own set of absolute coordinates, but rather as a "directional graph" of positions relative to each other and one single absolute "home position". This data structure brings a number of advantages over the existing organizational method of loose absolute coordinates:

1. More intuitive representation of the positions of each point, since a majority of trajectory waypoints are ultimately relative positions from some other point in the trajectory.
2. Consolidated data. Organizationally, having this data structure forces the good practice of keeping all point definitions within a single node/access point, which is essential for robot development work and any potential safety check methods we may want to implement in the future.
3. Ease of calibration. One of the most common causes for the robot being out of calibration is a reset of the robot encoders. Having all waypoints defined as a relative offset from a single absolute position can simplify the calibration process significantly, as there would only be the home position that needs to be updated through the homing procedure, and then all other points will dynamically update.
4. Human readability. When verifying dozens of waypoints, it is be easier for an operator to understand the spatial relation between all points rather than jus treading a list of absolute positions for each point.

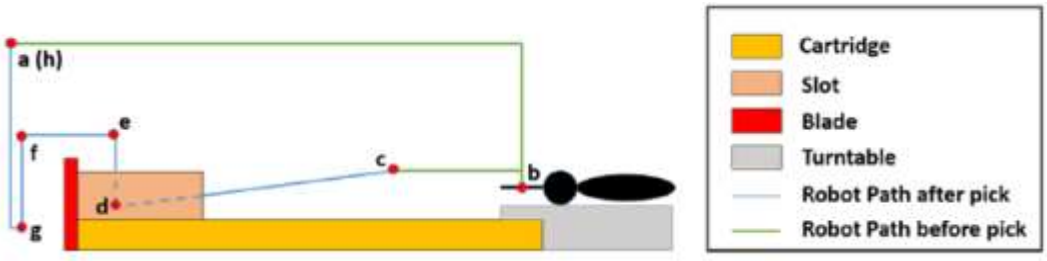

Figure 4: An example robot trajectory during operation. As shown, many waypoints along this trajectory are simple translations from other points in the trajectory. Relative positional representations of these points are more understandable.

The graph itself is stored as an array of “nodes” in a JSON file, where each “node” contains an identifying name, its parents (multiple since each node can have different parents for each axis), and the offset from its parents. The main reason for storing the structure in a JSON file is for direct human readability, as we want to give the operator the ability to directly edit the calibration values.

Upon system startup, a ROS node will load the calibration JSON file, and initialize a data graph. The node will have an endpoint for accessing the graph. A service requester will request the absolute coordinates of a given point by the name identifier, and the node will trace through the entire tree from the target node up to the root (Robot home) to retrieve the node’s absolute coordinates. For certain points that are not fixed for each operation (such as the mosquito’s proboscis location during pick), the node in the graph will remain in a partially defined state, and can only be retrieved once the requester provides more definition information (such as the image coordinates to be translated to X, Y coordinates through the handeye calibration).

The graph ROS node also has an additional endpoint implemented to “edit” points. This is used for the calibration of this graph. For the calibration of this graph, robot homing must be performed first/verified, since all other points depend on this root node. The calibration procedure can then calibrate for individual points in the graph by moving the robot to the target position in the workspace, reading the corresponding encoder counts, and updating the offset values from its parent nodes (as well as any downstream nodes).

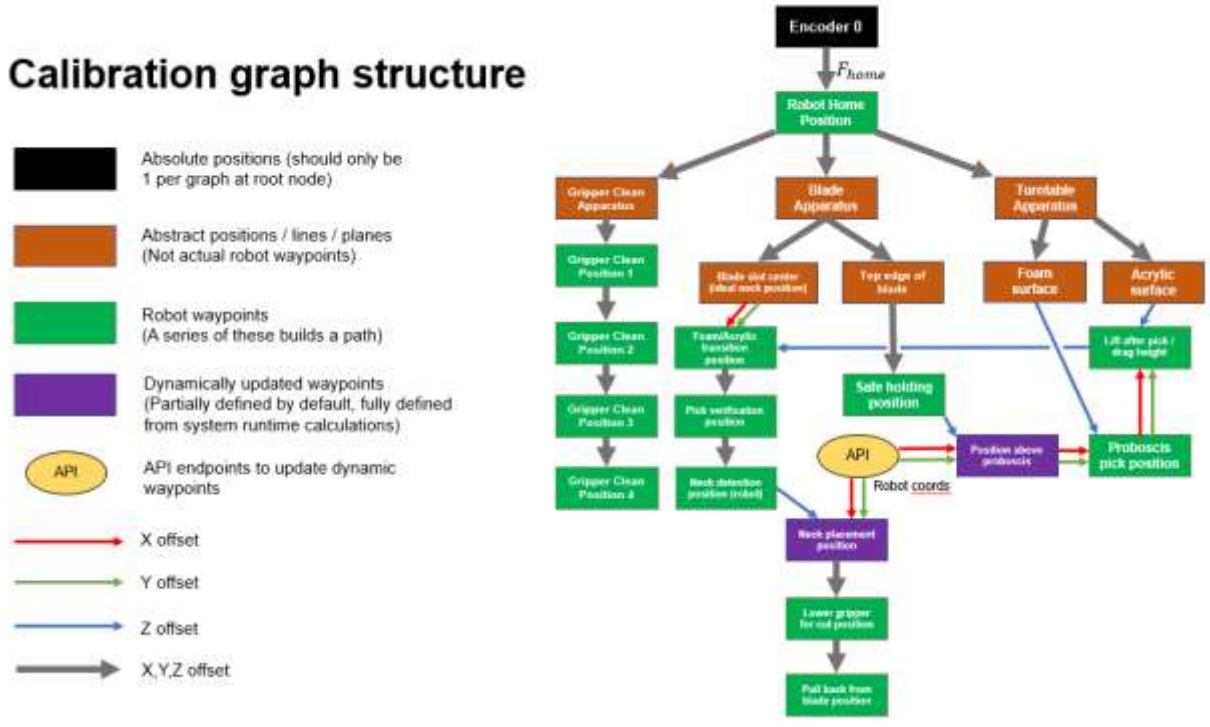

Figure 5: A graph representing the spatial dependencies of each of the robot's current trajectory waypoints

The prototype implementation of this graph was able to traverse and retrieve individual trajectory waypoints as intended. It is also able to correctly deny any requests for robot positions that are partially defined without any supplemental information.

Future Work – Robot Waypoint Calibration

The data structure itself is complete, but the calibration procedure still needs to be deployed and facilitated on a GUI, similar to the other procedures being developed.

Afterwards, this system will need to be integrated into the current robot operations, and validation tested with full mosquito processing workflows.

Technical Approach & Results – Handeye Calibration

The goal of the **hand-eye calibration** task is to effectively determine the registration mapping between the main overhead camera's frame to the robot's workspace. Prior to this project, the robot system already has an operational handeye calibration procedure. This procedure moves the robot to N points in a rectangular grid near the 2D plane where the mosquito manipulation occurs, detects the robot gripper tooltip position at each of these points, and fits the resulting point correspondences using a Bernstein polynomial fit. Coefficients for this polynomial fit is then saved in a config file. This config file is then opened and read by a dedicated ROS service node that can translate between robot and camera coordinates.

This previous implementation, however, was slow, required extensive operator input, and needed to be updated along with some of the robot system's newer software architecture refactoring. To address these issues, we completely refactored the previous handeye calibration script, moving it to a ROS node interfacing with a GUI, similar to that of the robot homing procedure. Our implementation also adds a number of much needed quality of life improvements for the calibration procedure, such as the ability to accept or deny a calibration after the main procedure is completed.

Our calibration procedure runs in approximately 4 minutes (previously 10 minutes), and requires only 2 operator inputs (one to start the procedure, and one to confirm/deny the new calibration), with the exception of a number of preparation steps.

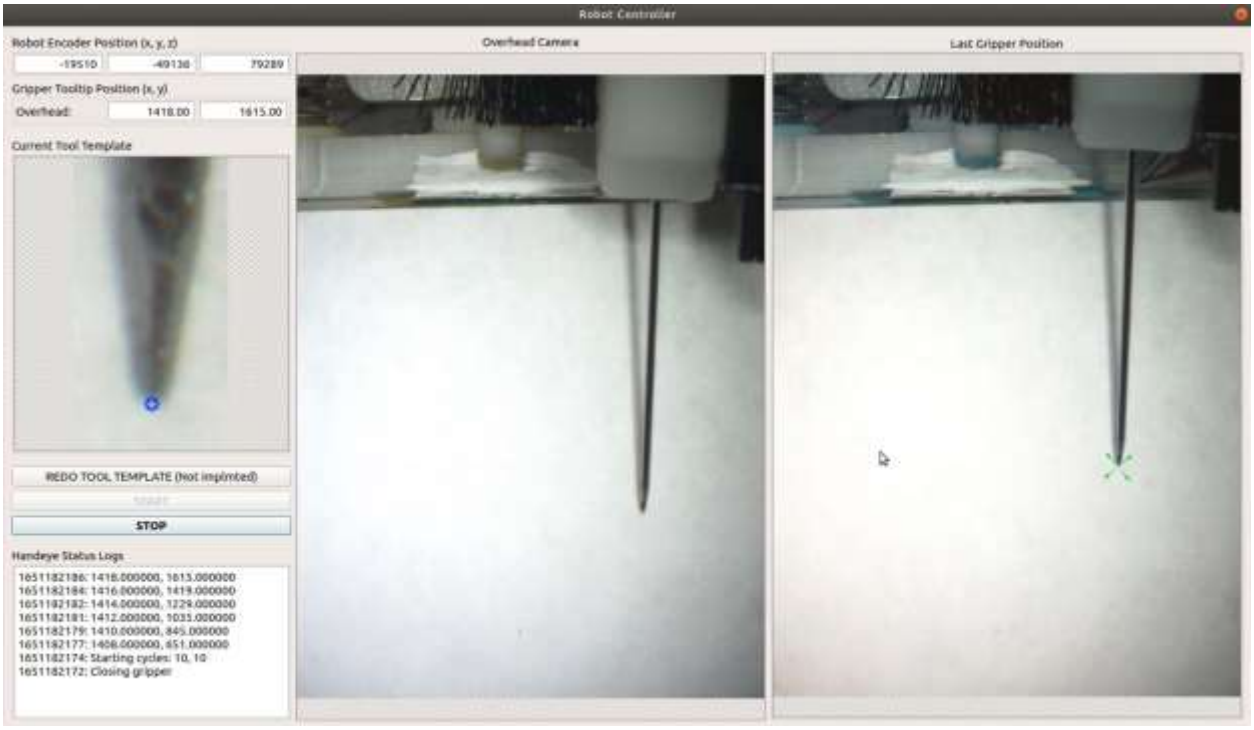

Figure 6: Handeye calibration GUI

Future Work – Handeye Calibration

There are still a few steps in the calibration procedure that can be further automated, such as the initial positioning of the tool at the correct starting position (related to the waypoint calibration component). Improving on these items can further improve the autonomous capacity of this procedure.

The gripper tooltip detection algorithm deployed in the calibration uses a traditional IoU method to match a tooltip image template to the actual image taken. This process has been reliable but requires the operator to manually obtain the tooltip image template every time the calibration procedure needs to be completed. This method also requires the background to be completely plain, which is achieved by placing a white piece of paper. The machine-learning based gripper tooltip detection algorithm being developed for the robot homing component can be deployed here to eliminate the need for this tool template and the white background, which can simplify this procedure significantly.

Management Summary

To summarize, this project was able to meet most of the expected deliverables established at the beginning of the project, and there is further work planned for the near future to tackle some of the remaining deliverables.

The minimum deliverables of this project were functional prototypes of each of the three calibration tasks. The expected deliverables were system verified and improved calibration workflows with GUIs for each workflow. The maximum deliverables were completely automated calibration workflows that require extremely little to no operator input beyond the initialization of the procedures, as well as extensive operator use feedback.

With the GUIs for the robot homing and handeye calibrations verified and deployed, we were able to meet the expected deliverables for those two tasks. However, we were not able to meet the expected deliverables for the waypoint calibration task, as there is currently only a prototype implementation of the structure without the GUI, and has yet to be tested within the full software system of the robot.

As for future work, there are a number of quality of life and usability improvements that should be made to each of the components, as discussed in the previous sections. There is also significant amount of work related to the integration of these calibration components in the greater robot development workflow, many of which touch upon components that are outside of the scope of this project, namely:

- Robot control methods (direct interface with the Galil vs. a single shared interface across all robot tasks)
- Improved computer vision performance, especially with gripper tooltip detection, as discussed previously
- Robot development and operational workflow, potentially including a complete robot operation GUI for the actual robot operation.
- Training future operators on using these calibration components, beyond just usage instructions.

As I will continue to contribute to this project over the coming summer, I plan on resolving many of these outstanding issues if the project priorities align. However, if time does not allow, I will ensure that the developments made with this project will be easy to expand on by any future contributors.

Over the course of this project, I was able to learn a significant amount about designing and implementing user-facing interfaces such as GUIs, and many considerations that need to go into designing these interfaces for its intended users. I also learned methods for developing software architecture that involves many distributed subcomponents (vision, hardware, data, etc.). From a project management perspective, I was able to expand upon my understanding of setting, tracking, and evaluating goals/progress for projects, as well as how to evaluate priorities when there is an overwhelming amount of work to be done for a single project.

References and Reading List

- H. Wu, J. Mu, T. Da, M. Xu, R. H. Taylor, I. Iordachita, and G. S. Chirikjian, "Multi-mosquito object detection and 2D pose estimation for automation of PfSPZ malaria vaccine production", in IEEE 15th International Conference on Automation Science and Engineering (CASE), Vancouver, BC, August 22-26, 2019. pp. 411-417.
- W. Li et al., "Automated Mosquito Salivary Gland Extractor for PfSPZ-based Malaria Vaccine Production," 2021 IEEE International Conference on Robotics and Automation (ICRA), 2021, pp. 866-872, doi: 10.1109/ICRA48506.2021.9560959.

Appendix

Code repositories:

All of the code for the robot, including the code developed as a part of this project, are stored in this repository group on GitLab:

https://git.lcsr.jhu.edu/mosquitoproject

The code repositories directly containing work covered by this project are:

https://git.lcsr.jhu.edu/mosquitoproject/rqt_sanaria_robot_controller

https://git.lcsr.jhu.edu/mosquitoproject/sanaria_calibration_tools

https://git.lcsr.jhu.edu/mosquitoproject/sanaria_transformation_graph

https://git.lcsr.jhu.edu/mosquitoproject/sanaria_handeye

Documentation on system design is in the documentation repository:

https://git.lcsr.jhu.edu/mosquitoproject/sanaria_docs]

Presentations and reports developed by this project are located on the course wiki page:

https://ciis.lcsr.jhu.edu/doku.php?id=courses%3A456%3A2022%3Aprojects%3A456-2022-01%3Aproject-01