Tool Gravity Compensation and Deflection Characterization for the Galen

Adam Polevoy
Department of Biomedical Engineering
Johns Hopkins University
apolevo1@jhu.edu

Parth Singh
Department of Biomedical Engineering
Johns Hopkins University
psingh21@jhu.edu

May 9, 2019

1 Objective

The objective of this project was to develop and validate models for tool gravity compensation and arm deflection for the Galen Surgical System. The gravity compensation model was to be implemented into the Galen code base for multiple tools. The arm deflection was to be modeled both analytically and with a neural network.

2 Technical Summary

2.1 Background

The Galen is a general purpose, hand-over-hand, admittance control robot developed mainly for otolaryngology. In hand-over-hand admittance control, the robot and surgeon both hold the tool. As the surgeon tries to move the tool, a force sensor picks up the applied force and torque and moves the tool proportionally. Thus, accurate force and torque measurement is key to the robot control scheme. It is currently being commercialized by Galen Robotics inc, but was originally developed as PhD project in the LCSR and is still the subject of many research projects. The Galen reads values at a force/torque sensor and calculates the expected force/torque values at the user's point of contact on a tool. The tool then moves proportionally to the force exerted, allowing a surgeon to use a tool with the additional benefits of stability, tremor cancellation, and virtual fixtures.

2.2 Problem

The force sensor used by the robot has no way to distinguish between the forces/torques exerted by the surgeon and those exerted by the tool itself. Currently, this problem is solved by re-biasing (zeroing) the force sensor in the home configuration once the tool is attached. However, rotating the tool results in a different center of mass position with respect to the force sensor. This causes new force/torque readings, which act as an unwanted input into the robot control system. This can cause the tool to drift when the surgeon is not holding it and can act against the surgeon's intended motions. This can result damage to sensitive structures near the operating area. Additionally, it is tiring for the surgeons that have to fight against the weight of the tool. The system also suffers from deflection issues. When force is applied to the robotic arm, the true position of the robot varies from the expected position because of flexibility in the joints. This can cause control issues, especially when trying to use virtual fixtures.

2.3 Approach

2.3.1 Parameter Estimation

Gravity compensation depends on the weight and the center of mass (CoM) of each tool. Additionally, the weight and CoM that affects the force sensor when no tool is attached, referred to as the bias, must be taken into account. These parameters had to be regressed for effective compensation.

To regress bias weight and CoM, the force sensor was rebiased (zeroed) in it's home position, and then the tool was attached. A behavior was written to move the robot to 56 configurations, wait until motion has stopped at each configuration, and collect force sensor data. Using the data from these 56 configurations and the following equations, the bias weight and CoM was regressed.

$$A_1 = \begin{bmatrix} R_{ts}(\vec{\theta}^{(1)})^T - R_{ts}(\vec{0})^T \\ ... \\ R_{ts}(\vec{\theta}^{(56)})^T - R_{ts}(\vec{0})^T \end{bmatrix}, \ Y_1 = \begin{bmatrix} f_s^{(1)} \\ ... \\ f_s^{(56)} \end{bmatrix} \tag{1}$$

$$f_b = (A_1^T A_1)^{-1} A_1^T Y_1 \tag{2}$$

$$A_2 = \begin{bmatrix} (R_{ts}(\vec{\theta}^{(1)})^T - R_{ts}(\vec{0})^T)\widehat{f}_b(-R_{ts}(\vec{\theta}^{(1)})) \\ ... \\ (R_{ts}(\vec{\theta}^{(56)})^T - R_{ts}(\vec{0})^T)\widehat{f}_b(-R_{ts}(\vec{\theta}^{(56)})) \end{bmatrix}, \ Y_2 = \begin{bmatrix} \tau_s^{(1)} \\ ... \\ \tau_s^{(56)} \end{bmatrix} \tag{3}$$

$$p_{sb} = (A_2^T A_2)^{-1} A_2^T Y_2 \tag{4}$$

Similarly, the tool weight and CoM can be regressed using the following equations.

$$A_3 = \begin{bmatrix} R_{ts}(\vec{\theta}^{(1)})^T \\ ... \\ R_{ts}(\vec{\theta}^{(56)})^T \end{bmatrix}, \ Y_3 = \begin{bmatrix} f_s^{(1)} - (R_{ts}(\vec{\theta}^{(1)})^T - R_{ts}(\vec{0})^T)f_b \\ ... \\ f_s^{(56)} - (R_{ts}(\vec{\theta}^{(56)})^T - R_{ts}(\vec{0})^T)f_b \end{bmatrix} \tag{5}$$

$$f_t = (A_3^T A_3)^{-1} A_3^T Y_3 \tag{6}$$

$$A_4 = \begin{bmatrix} -R_{ts}(\vec{\theta}^{(1)})^T \widehat{f}_t R_{ts}(\vec{\theta}^{(1)}) \\ ... \\ -R_{ts}(\vec{\theta}^{(56)})^T \widehat{f}_t R_{ts}(\vec{\theta}^{(56)}) \end{bmatrix}, \ Y_4 = \begin{bmatrix} \tau_s^{(1)} - (R_{ts}(\vec{\theta}^{(1)})^T - R_{ts}(\vec{0})^T)\widehat{f}_b(-R_{ts}(\vec{\theta}^{(1)}))p_{sb} \\ ... \\ \tau_s^{(56)} - (R_{ts}(\vec{\theta}^{(56)})^T - R_{ts}(\vec{0})^T)\widehat{f}_b(-R_{ts}(\vec{\theta}^{(56)}))p_{sb} \end{bmatrix} \tag{7}$$

$$p_{st} = (A_4^T A_4)^{-1} A_4^T Y_4 \tag{8}$$

2.3.2 Compensation

Given the mass and center of mass parameters, the expected force sensor readings due to tool and bias gravity can be calculated using the following equations. The expected force sensor readings can then be subtracted from the actual force sensor readings to yield compensated values.

$$g(\vec{\theta}^{(i)}) = (R(\vec{\theta}^{(i)}), p) \tag{9}$$

$$Ad_{g(\vec{\theta}^{(i)})} = \begin{bmatrix} R(\vec{\theta}^{(i)}) & \widehat{p}R(\vec{\theta}^{(i)}) \\ 0_{3x3} & R(\vec{\theta}^{(i)}) \end{bmatrix} \tag{10}$$

$$F_{s,prediction} = Ad^T_{g_{ts}(\vec{\theta}^{(i)})} F_t + Ad^T_{g_{bs}(\vec{\theta}^{(i)})} F_b - Ad^T_{g_{bs}(\vec{0})} F_b \tag{11}$$

$$F_{s,compensated} = F_{s,uncompensated} - F_{s,prediction} \tag{12}$$

2.3.3 Implementation

The above analysis was first performed in MATLAB to validate the regression and compensation algorithm. However, once the estimation and calibration was validated, it was implemented into the Galen code base. This implementation consists of two parts: parameter estimation and real time compensation.

A robot behavior was implemented for parameter estimation. This behavior moves the the robot to 56 configurations and stores the average force sensor readings at each configuration. It then performs parameter estimation as described in subsection 2.3.1 and writes the mass and CoM parameters to the tool's parameter Json file.

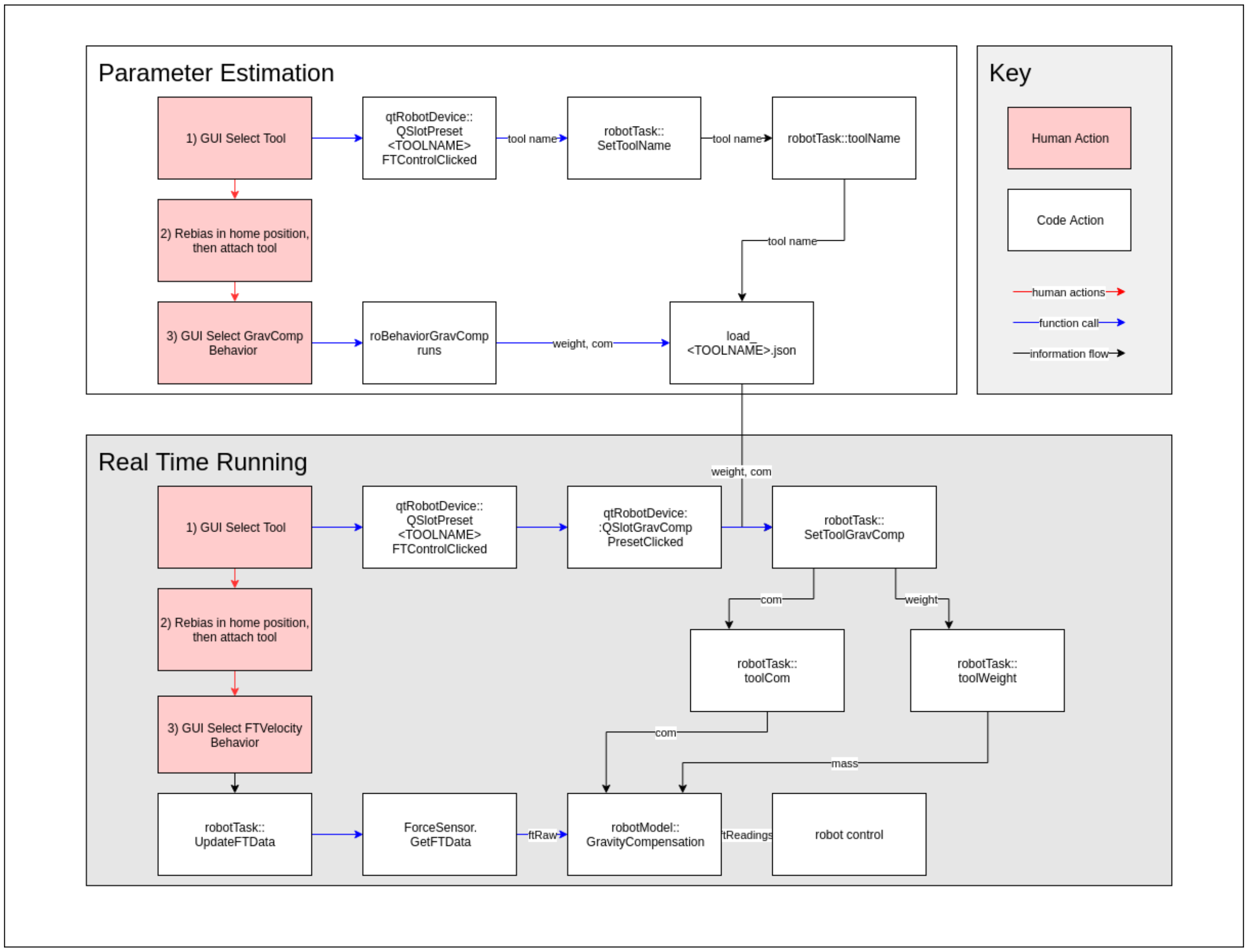

Figure 1: Workflow of Code Implementation

A function was implemented to perform gravity compensation. It loads in the tool weight and CoM from the Json file, and then performs gravity compensation as described in subsection 2.3.2

Our gravity compensation code has been connected to several buttons in the existing Galen GUI. When a tool button is selected, the tool name and parameters are loaded. The behavior used for parameter estimation was added to the behavior drop down. Furthermore, gravity compensation can be toggled on or off using a gravity compensation toggle button.

Documenation for the code can be found in section 4. Figure 1 is a block diagram the depicts the workflow of the code implementation.

2.3.4 Deflection Data Collection and Preprocessing

In order to predict deflection, we had to compare the true position of the end effector compared to the position expected by the robot. An optical tracker was used to find the position and orientation of the end effector and a fiducial in the tracker frame, $g_T t$ and $g_T f$. The robot forward kinematics was used to find the position and orientation of the end effector in the base frame, $g_B t$. The frame setup is shown in Figure 2. For the first few seconds of each data collection, no force was applied to the end effector. Assuming there is zero deflection when no force is being applied, these first few seconds of data collection can be used for calibration. The frame transformation between the fiducial and the robot base can be calculated.

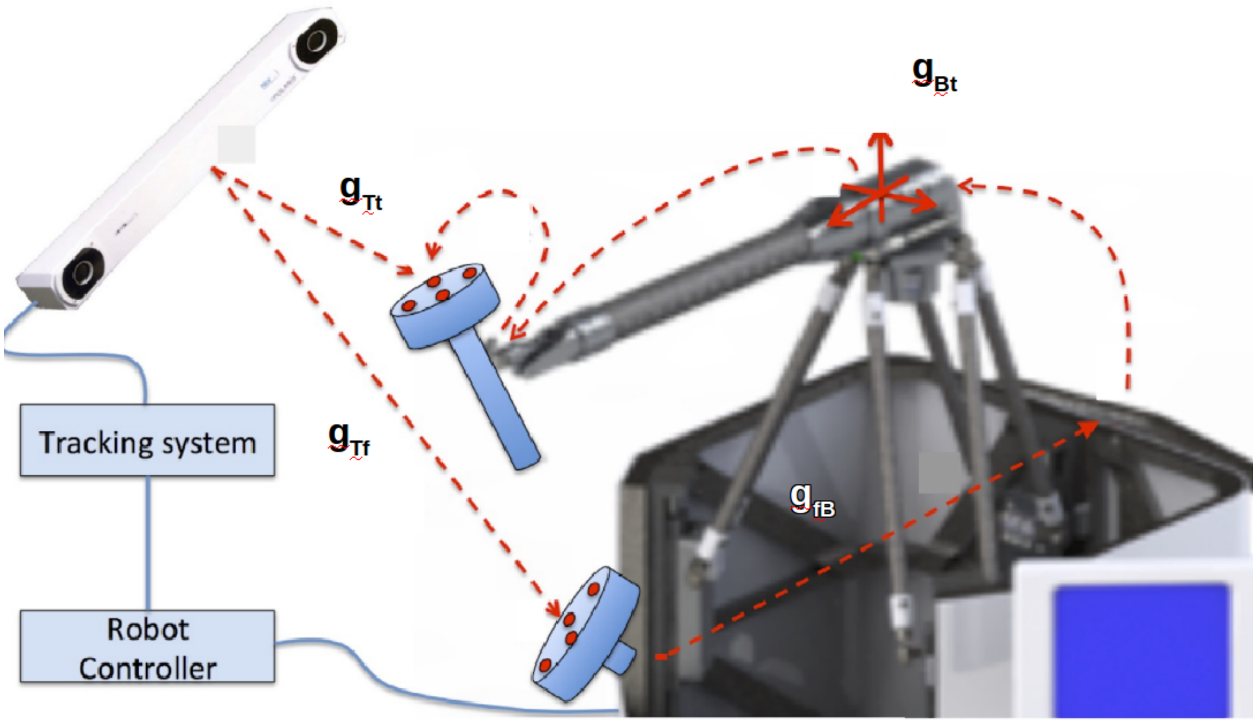

Figure 2: Image Credit: CIS 1 2018 homework 4

$$g_{fB} = (g_{Tf}^{-1})(g_{Tt})(g_{Bt}^{-1}) \tag{13}$$

Given the position and orientation of the tool in the tracker coordinate frame, we can calculate the true position and orientation of the tool in the base frame at each time point.

$$g_{Bt_{true}} = (g_{fB}^{-1})(g_{Tf}^{-1})p_{Tt} \tag{14}$$

To collect data, we moved the robot to various positions within the configuration space and applied several levels of forces in different directions along all axes, recording force sensor data, joint angles, the transformation from the base reported by the robot, and the transformation between the fiducial and tool given by the optical tracker. We then find the true position of the tool with respect to the base using the above equation.

We collected 8 different data sets, with two at any particular position and 2 additional calibration sets to ensure the robot position remained constant over time.

2.3.5 Deflection Analytical Approach

As a first attempt, independence of axes was assumed, and deflection was modeled linearly with respect to force applied in each axis. Figure 3 shows true and predicted deflection when the training and testing sets have the same joint configurations. The two main shortfalls are that is is unable to predict the z-axis deflection and it is unable to predict the hysteresis that is clearly present in the system. After a large force, there seems to be a residual deflection that remains after the force is gone. Figure 4 shows true and predicted deflection when the training and testing sets have different joint configurations. Clearly, the predictions are worse, so stiffness of the arm must depend on joint configuration.

A second attempt was to not assume independence of joint axes. In this case, deflection of each axis was modeled as a linear combination of force in each axis, determined by linear least squares. Figure 5 shows that while this allows for more accurate prediction of z-axis deflection, it still was unable to predict the hysteresis of the system. Additionally, no linear relationship between joint values and arm stiffness was found.

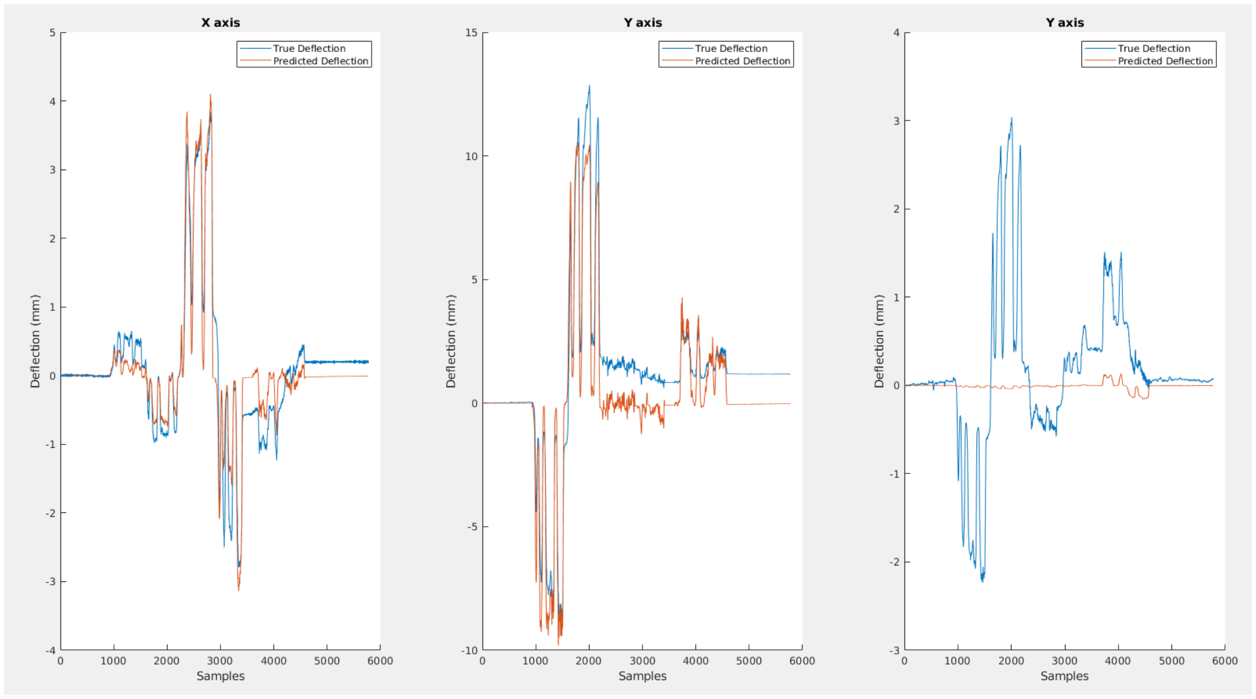

Figure 3: Deflection results, training and testing sets have same joint configuration

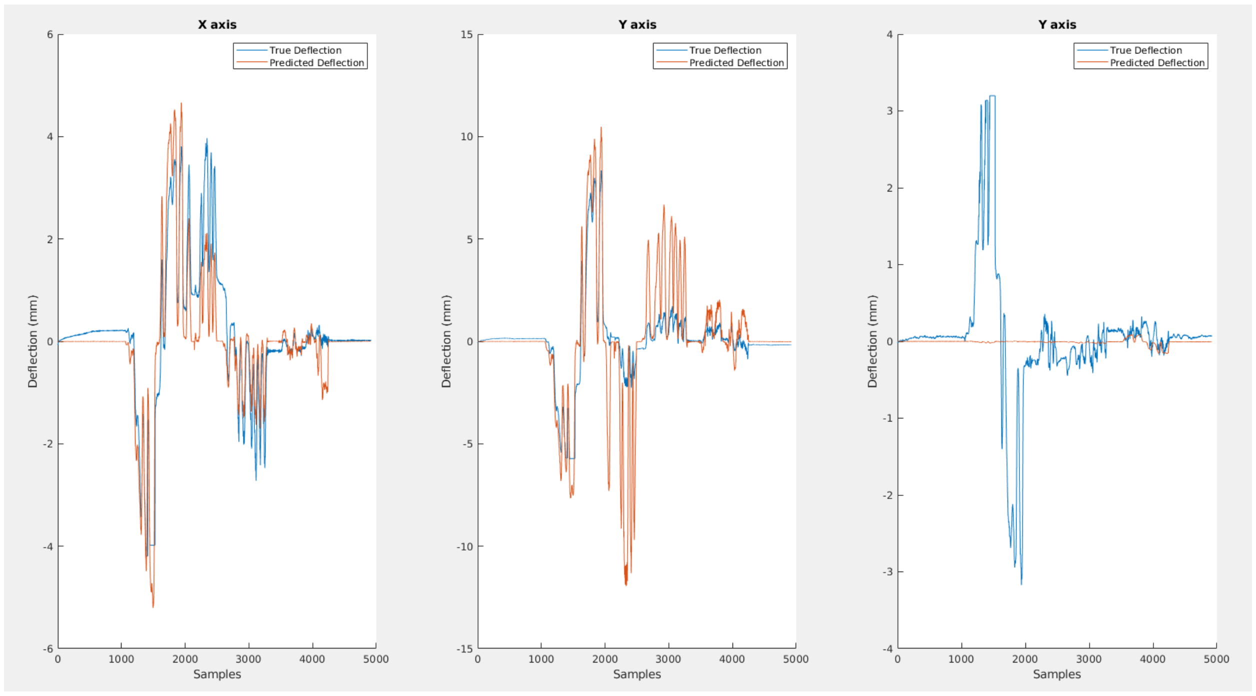

Figure 4: Deflection results, training and testing sets have different joint configuration

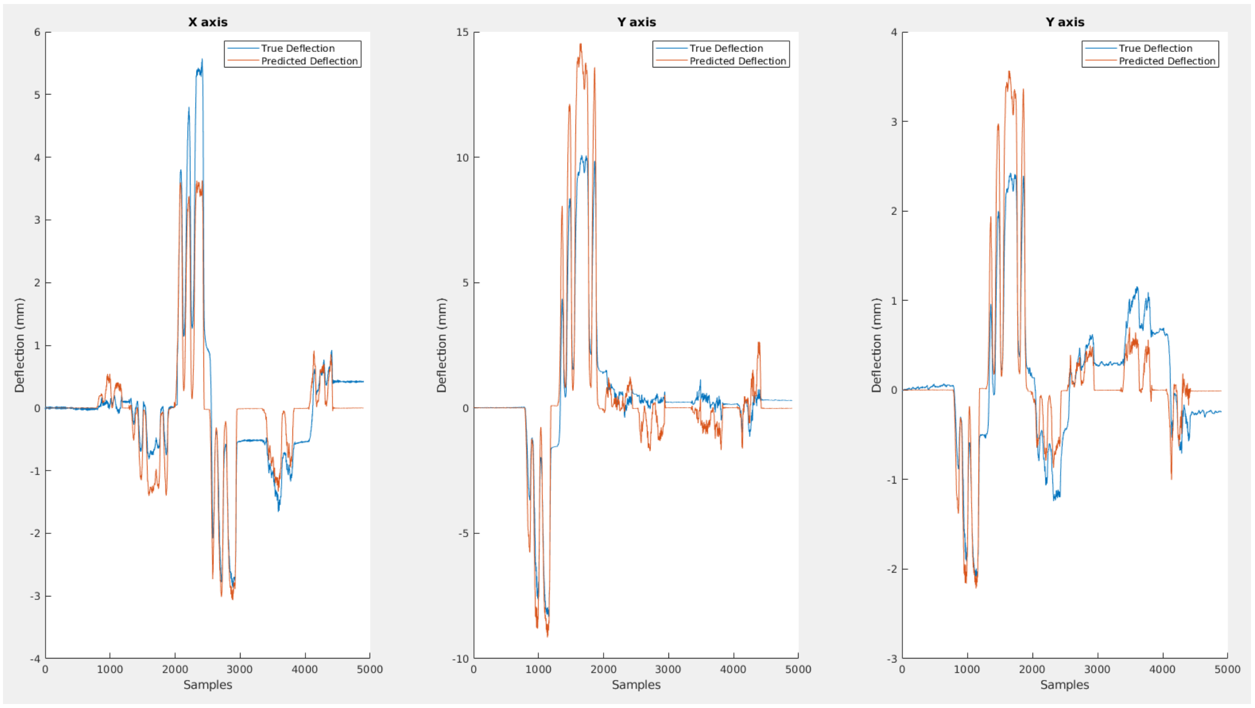

Figure 5: Deflection results, not assuming independence

2.3.6 Deflection Machine Learning Approach

After finding nonlinearities and hysteresis in the data that was collected for deflection, it was decided a machine learning approach may be more effective. A long short-term memory recurrent neural network in Matlab was implemented in Matlab, as this is an effective model for systems with nonlinearities and hysteresis. There were several different attempts with different hyperparameters, but the best results came from the following configuration. We used a single layer LSTM network with 18 hidden units, a dropout rate of 0.05, and 150 iterations. We used 6 sets of our data for training and verified on the remaining 2 sets. A visual representation of our network structure can be seen in Figure 6. The results of training the network can be seen in Figure 7.

Given more time, this model could be much improved. Primarily, significantly more data must be collected to train the network on. Additionally, the regression layer and fully connected layers may be removed. This will help significantly with overfitting and would make the model more generalizable. Additionally, data of a longer time length with more variability should be collected to train the network on.

2.4 Results

2.4.1 Gravity Compensation

Initial results of the gravity compensation data analysis were positive. For two separate tools, parameters were estimated from training data sets. On separate testing data sets, gravity compensation was performed. If gravity compensation worked, the expected compensated force/torque values would be near zero. As shown in figures 8 and 9, this was the case.

Once the algorithm had been proven effective, it was integrated into the Galen. Gravity compensation completely eliminated tool drift that had occurred when the tool was moved from the re-biasing location. Furthermore, it has been reported that the gravity compensation improved the transparency of control. In other words, users felt that the robot moved more cooperatively with them and resisted them less.

2.4.2 Deflection Characterization

We created a LSTM model to predict deflection to varying levels of success. We managed to reduce our Root Mean Square Error to as low as 9.483. We can likely increase the accuracy of our model by collecting significantly more data,

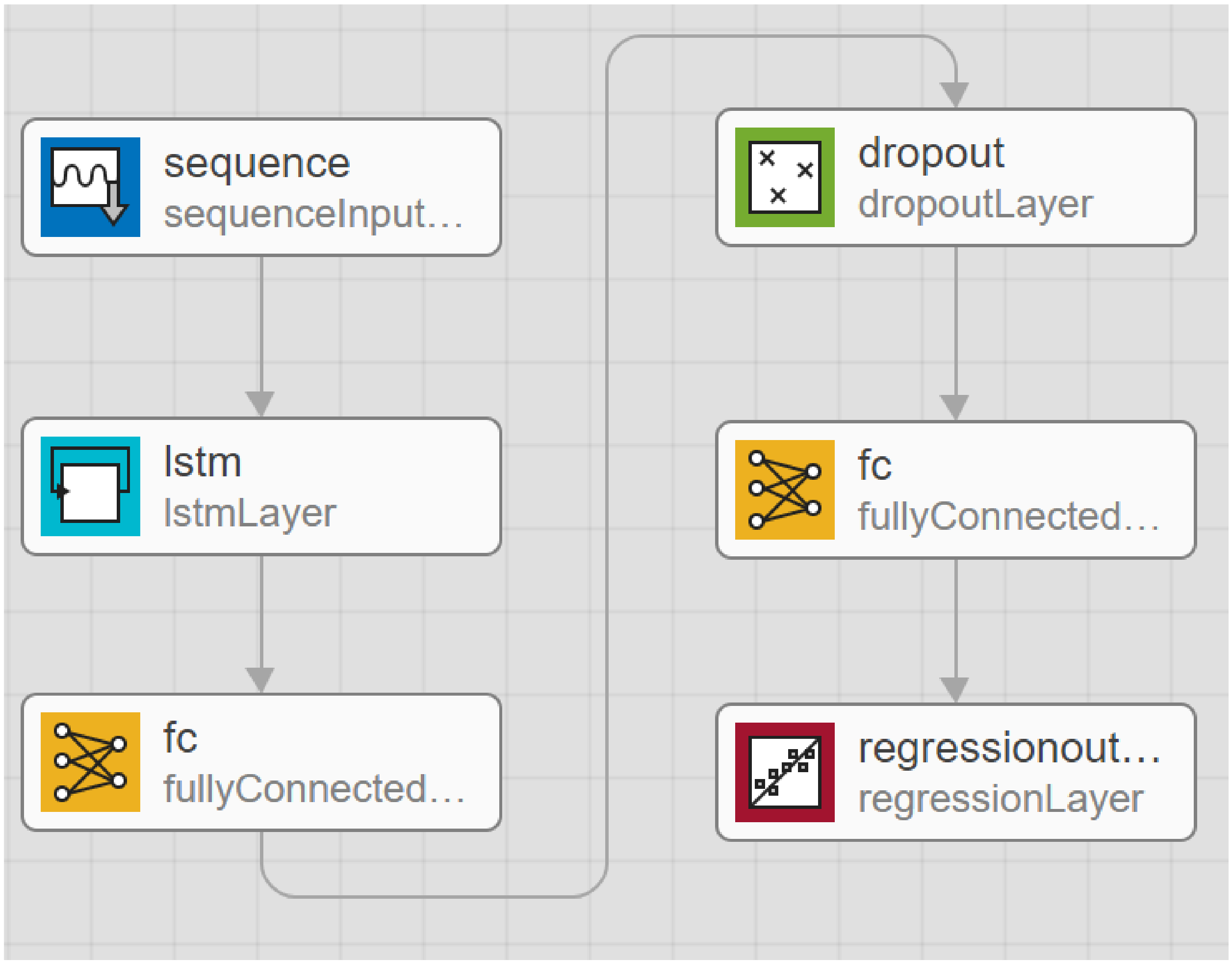

Figure 6: Recurrent Neural Network Structure

Table 1: Gravity Compensation Pointer Tool Error

Force Sensor Axis	RMSE (N and Nmm)
Fx	0.0214
Fy	0.0174
Fz	0.0306
Tx	0.2835
Ty	0.6009
Tz	0.3330

Table 2: Gravity Compensation Retractor Tool Error

Force Sensor Axis	RMSE (N and Nmm)
Fx	0.0245
Fy	0.0134
Fz	0.0388
Tx	0.6621
Ty	0.7079
Tz	0.3997

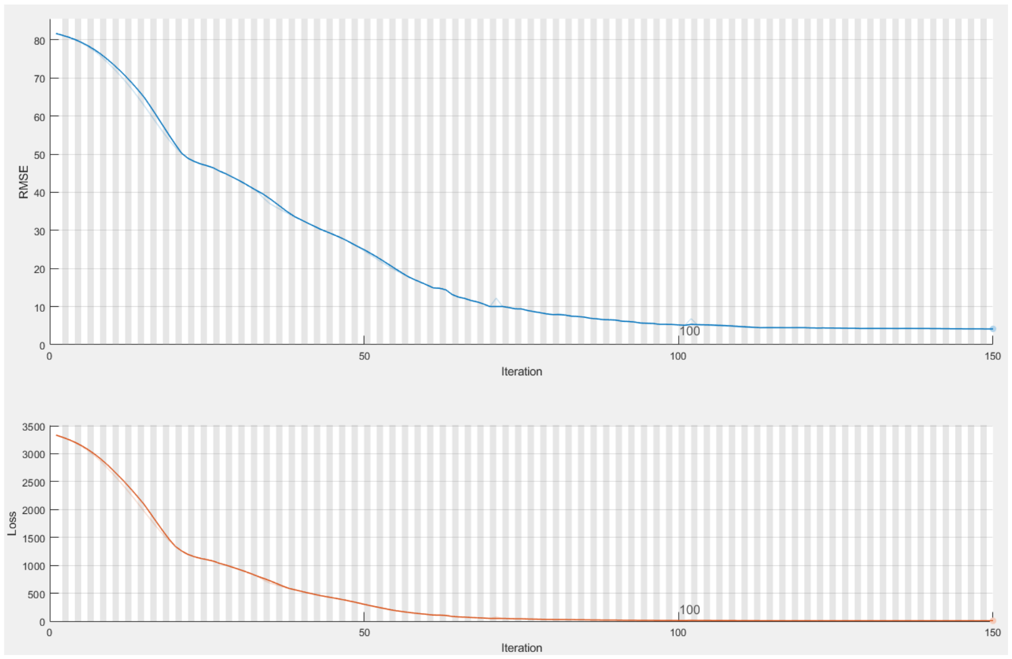

Figure 7: RNN Training results

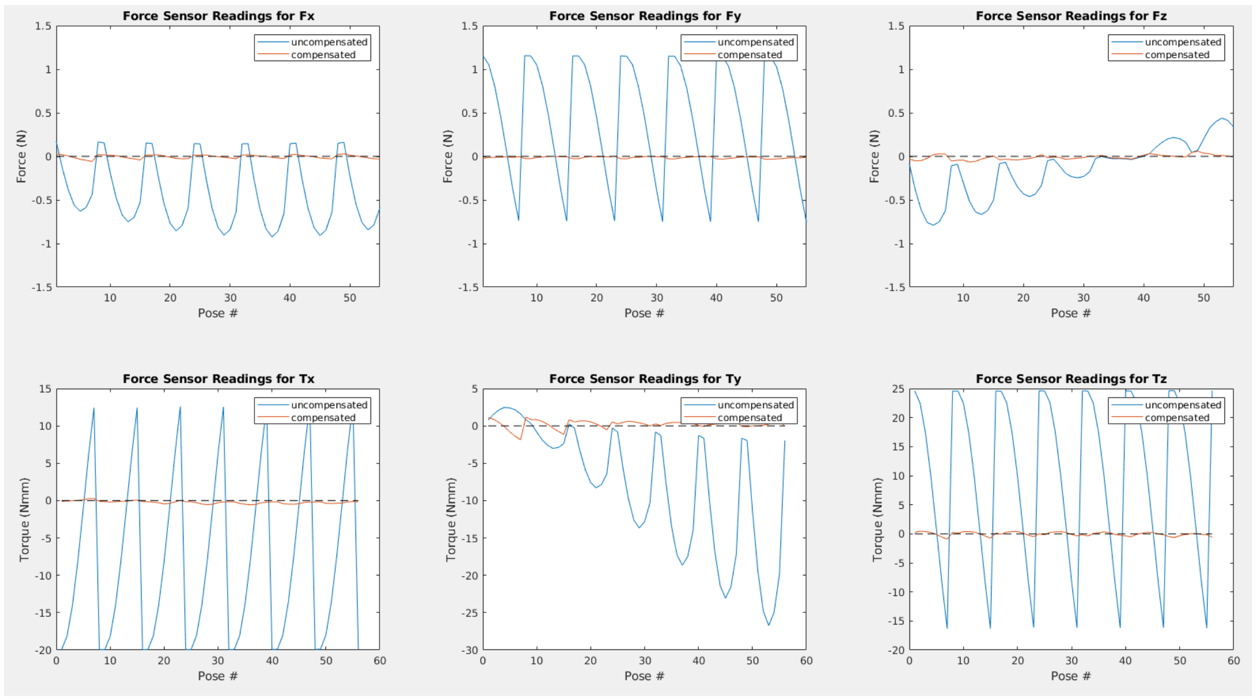

Figure 8: Gravity Compensation for Pointer Tool

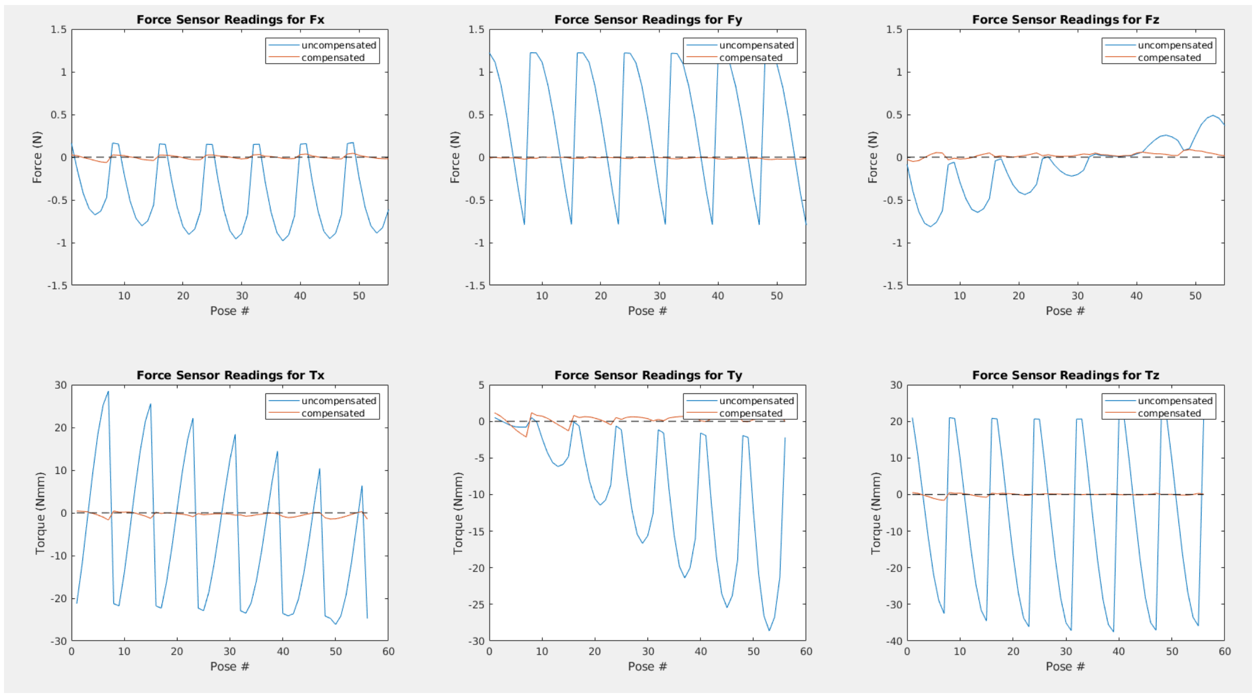

Figure 9: Gravity Compensation for Retractor Tool

in more varied joint positions. In addition, the issue of hysteresis we found was still prevalent in our model's prediction, which can be seen in Figures 10 and 11.

2.5 Significance

Both gravity compensation and deflection characterization are pivotal to the commercial success of the Galen micro-surgical system. Without gravity compensation implemented, the Galen system may move under the weight of the tool it is holding and cause permanent damage to any patient that is being operated on. In invasive procedures, especially on small target areas, even the slightest movement can result in patient injury. It also removes the strain that surgeons face from holding tools for the duration of surgery. Deflection characterization is also important, particularly for the future implementation of virtual fixtures. Virtual fixtures are areas defined by software that the robot is not allowed to move through. This can be used to prevent the risk of a surgeon cutting into an unwanted area and harming the patient. However, if the robot cannot accurate calculate its position, it could easily move into a virtual fixture and cause damage. Thus, our implementation of gravity compensation and the discoveries we've made about Galen arm deflection will have a very large impact on the success of the Galen micro-surgical system

3 Management Summary

3.1 Work Distribution

This project had several components, including developing data collection code, collecting data, analyzing data, developing models, testing models, integrating models. Adam focused on the development and analysis of data while Parth focused on the production and integration of code. However; most of the analysis and integration was done cooperatively.

3.2 Work Planned vs. Accomplished

Initially, we planned on gravity compensation for both the static cases (not taking into account forces due to motion) and dynamic cases. We also expected to have a working deflection model. We successfully integrated static gravity compensation abilities into the code, along with a complete workflow to calculate the parameters of any arbitrary tool. We, however, under the advise of our mentor, decided to focus on deflection characterization instead of dynamic

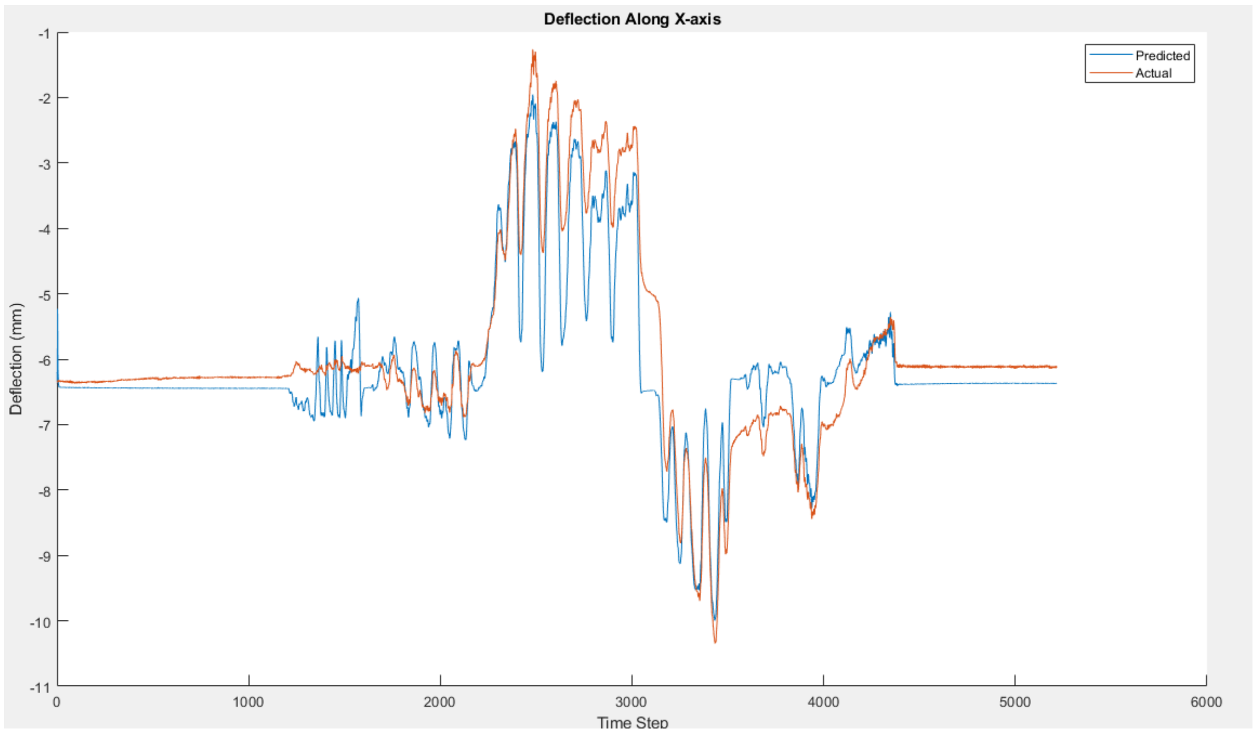

Figure 10: Results Example 1: Predicted vs actual deflection along X-axis

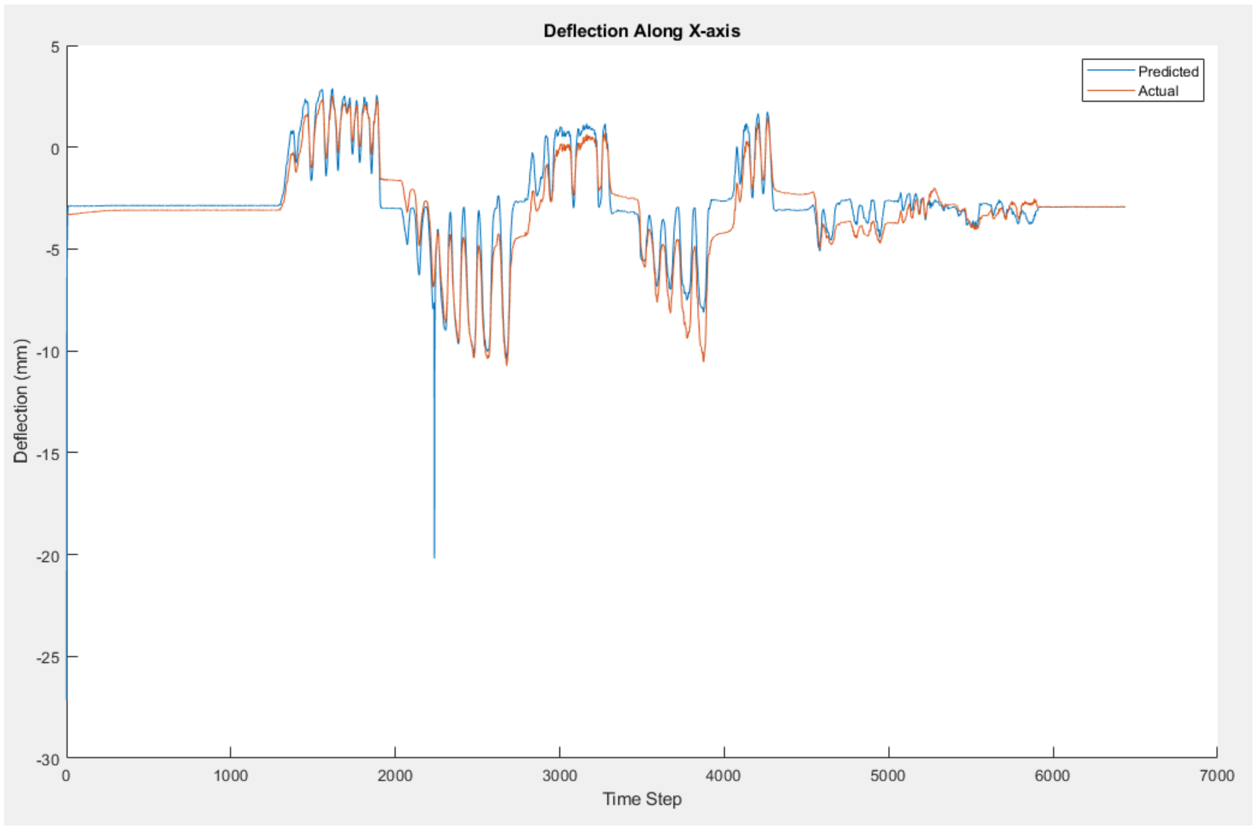

Figure 11: Results Example 2: Predicted vs actual deflection along X-axis

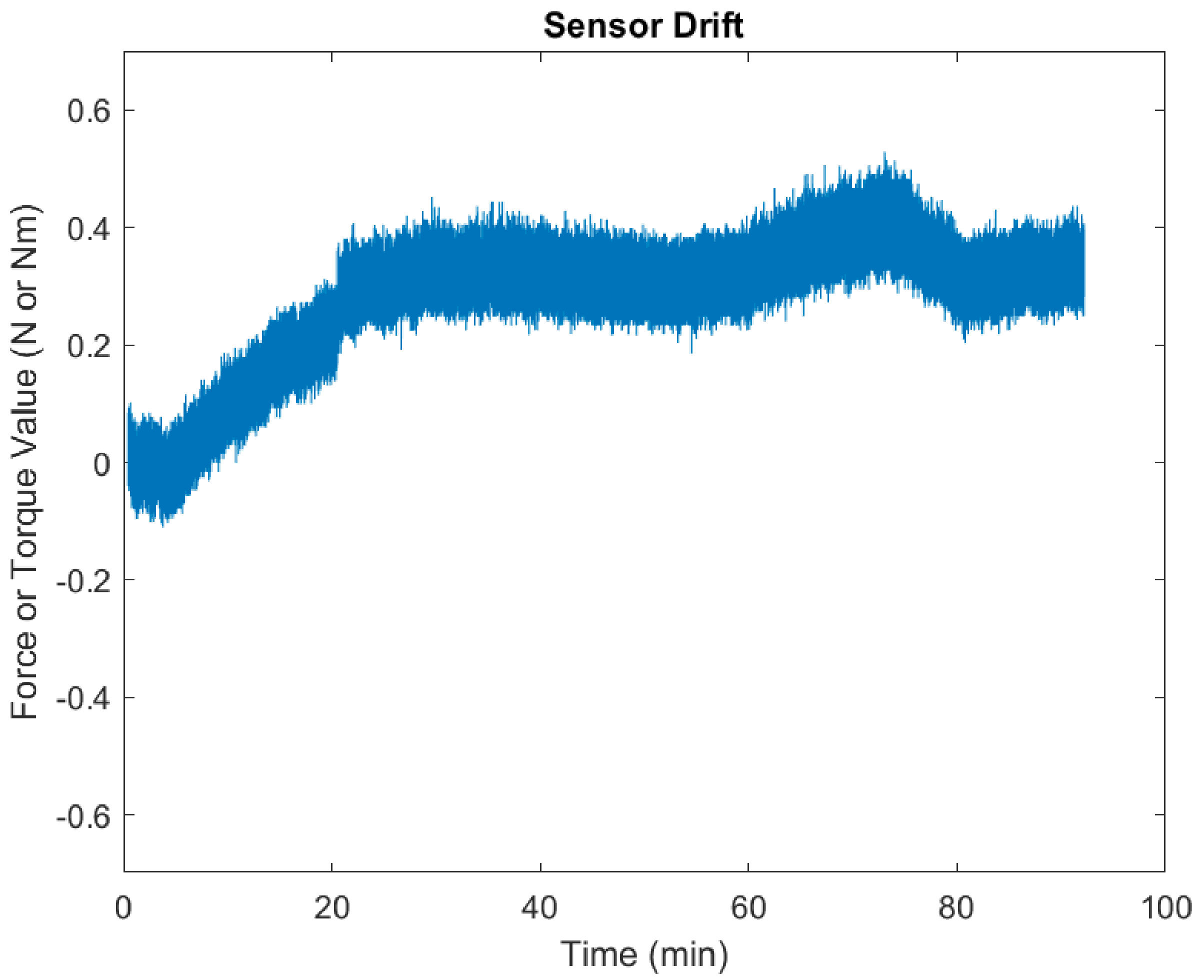

Figure 12: Force Sensor Drift in Galen Mk2

compensation. The forces caused by dynamic cases would not be very significant compared to the static cases.

We worked towards developing a model for deflection characterization, but we were unable to create a fully functional model. We initially wanted to approach it from an analytical perspective. However; after we discovered the issue of hysteresis and possible nonlinearities, we pivoted to a machine learning approach, using a RNN capable of modeling time-dependent data. While we developed a model, it does not work very well, so there is still much to be desired and further refinements can be made.

There were couple of delaying factors that prevented us from having more time to refine the deflection model. One of these factors was sensor drift on the Galen Mk2, the most advanced Galen microsurgical system. We found, while trying to figure out why our gravity compensation algorithm was not working, that the z-axis of the Mk2 had significant force drift. This drift can be seen in Figure 12. We collected force sensor drift data for 72 hours to see if the pattern was repeatable enough to model. However; we found that the behaviour was not predictable enought to model. Additionally, the kinematics of the robot due to the tilt joint was incorrect. As a result, we spent much time debugging our gravity compensation algorithm, when the problem was actually the robot itself. A plot of our gravity compensation results due to this kinematics error can be seen in Figure 13.

Both of these factors delayed us significantly, which caused us to spend more time doing static gravity compensation than initially expected.

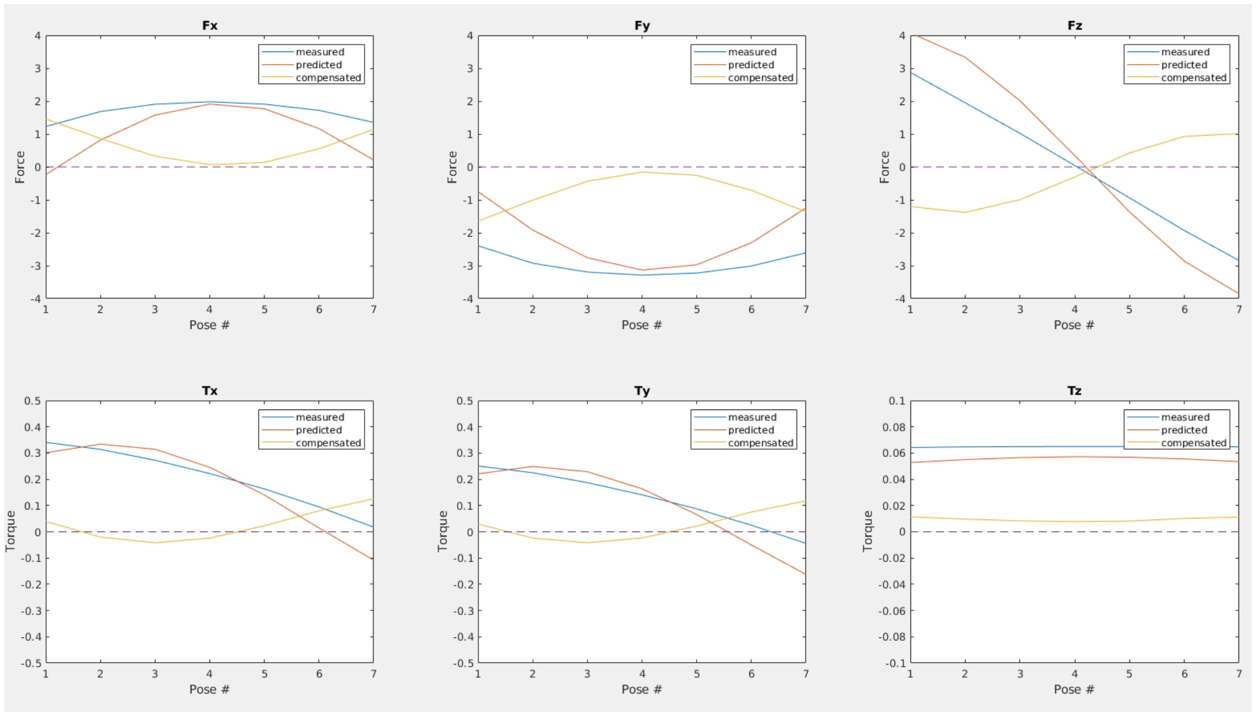

Figure 13: Gravity Compensation Results Due to Kinematic Error

3.3 Future Work

In the future, gravity compensation for dynamic tools will needed to be implemented, along with some way to predict the force generated by tools that are affected by cable drag. Deflection characterization will also need to be refined before it can be implemented in the Galen codebase. These tasks will likely fall on Parth, who will be working full time for Galen Robotics starting in August.

3.4 Lessons Learned

We faced several problems during the course of this semester. There were issues getting acquainted to the codebase. A majority of the Galen code was written by a single individual and as a result, was difficult to parse through and understand. There was a particular issue where we were operating under the assumption that a variable held the raw force values from the sensor but found that it was being processed elsewhere in the code. We learned how to navigate the codebase efficiently and trace though to find any issues. This taught us the importance of becoming completely familiar with the code before developing on top of it. We also learned the importance of having well documented code.

We also ran into issues where our deflection data collection was interrupted and there were jumps in the timestamp. This taught us that we should ensure that we carefully analyze data that we collect, and then adjust our data collection methods according before moving on to fitting models to the data. We learned that its better to look for patterns in small data sets before collecting larger ones.

4 Technical Appendices

Table 3 is a list of variables used in section 2.3.

Our code can be found in the GravityCompensationmk1 and GravityCompensation branches in the Galen research repo. Our model for deflection can be found in *tell me where to upload this code*

The following is a list of classes and methods that we implemented for gravity compensation

Classes:

Table 3: Interpretation of Variables

Variable	Interpretation
t	frame at CoM of tool, parallel to base
b	frame at CoM of bias, parallel to base
s	force sensor frame
T	tracker frame
B	base frame
$\vec{\theta}^{(i)}$	joint angles at configuration i
$R_{ts}(\vec{\theta}^{(i)})$	rotation between tool frame and sensor frame
$p_{sb},\ p_{sb}$	translation between sensor frame and tool/bias frame
$f_s^{(i)}$	force read by force sensor at configuration i
$\tau_s^{(i)}$	torque read by force sensor at configuration i
$f_t,\ f_b$	weight vector of tool/bias
F	wrench vector: $[f^T\ \tau^T]^T$
$\widehat{p}$	skew matrix of vector p

- roBehaviorGravComp: Runs gravity compensation parameter estimation for a tools
  - Called when gravity compensation behavior is selected in GUI
  - Note: Standard roBehaviorBase methods/properties not listed
  - Methods:
    - GCisMoving: Checks if the robot is moving
      - Outputs:
      - bool moving: True if moving, false if not
    - CalcToolGravComp: Regresses tool parameters (mass & com)
      - Inputs:
      - DataMat dM: data matrix of angles and force readings at poses
    - rotXEigen: creates X rotation matrix using Eigen
      - Inputs:
      - double theta: rotation angle
      - Outputs:
      - Eigen::Matrix3d rotx: 3x3 rotation matrix
    - rotYEigen: creates Y rotation matrix using Eigen
      - Inputs:
      - double theta: rotation angle
      - Outputs:
      - Eigen::Matrix3d roy: 3x3 rotation matrix
    - rotZEigen: creates Z rotation matrix using Eigen
      - Inputs:
      - double theta: rotation angle
      - Outputs:
      - Eigen::Matrix3d rotz: 3x3 rotation matrix
    - skewEigen: creates skew matrix using Eigen
      - Inputs:
      - Eigen::Vector3d v: 3x1 vector
      - Outputs:
      - Eigen::Matrix3d m: 3x3 skew matrix
  - Properties
    - std::vector<double> tilts: vector of tilt angles to iterate through
    - std::vector<double> rolls: vector of roll angles to iterate through
    - int tCntr: tilt index we are currently at
    - int rCntr: roll index we are currently at
    - int itersT: number of tilt iterations

    * int itersR: number of roll iterations
    * int waitCntr: used to hold robot at position while collecting data
    * int waitLimit: number of cycles each position is held for
    * int cyclesB4Recording: number of cycles waited before collecting data
    * DataMat dM: used to hold collected data from each pose
    * XMassMat X: constructed from joint angles for mass regression
    * XComMat X2: constructed from joint angles for com regression
    * YMat Y: constructed from force/torque readings, for mass and com regression
    * Xhome: home X position (delta platform)
    * Yhome: home Y position (delta platform)
    * Zhome: home Z position (delta platform)
    * Rhome: home roll angle
    * Thome: home tilt angle
    * rstart: start of range of roll angles
    * tstart: start of range of tilt angles
    * rend: end of range of roll angles
    * tend: end of range of tilt angles

Methods:

- robotModel::GravityCompensation
  - GravityCompensation: Performs gravity compensation on the raw force/torque sensor readings
  - Inputs:
    * vct6 FTreadings: The raw force/torque values (N and Nmm?)
    * const double roll: The roll joint angle (radians)
    * const double tilt: The tilt joint angle (radians)
    * const double toolWeight: The center of mass of the tool (mm?)
  - Outputs:
    * vct6 FTReadingsComp: The compensated force/torque values (N and Nmm?)
  - Called by robotTask::UpdateFTData
- robotModel::AdjointMat
  - AdjointMat: Creates an adjoint matrix from a given rotation and translation
  - Inputs:
    * const vct3x3 rotMat: A rotation matrix
    * const vct3 transVct: A translation vector
  - Outputs:
    * vctFixedSizeMatrix<double, 6, 6, VCT_ROW_MAJOR> adjointMat: An adjoint matrix
  - Called by robotModel::GravityCompensation
- robotTask::SetToolGravComp
  - SetToolGravComp: sets the tool parameters for gravity compensation
  - Inputs:
    * const mtsDoubleVec mtoolgravcomp: Tool parameters for gravity compensation
  - Called by qtRobotDevice::QSlotGravCompPresetClicked
- qtRobotDevice::QSlotGravCompPresetClicked
  - QSlotGravCompPresetClicked: Reads json file and extracts tool grav comp parameters
  - Inputs:
    * const QString &: The name of the tool
  - Called by qtRobotDevice::QSlotPreset<TOOLNAME>FTControlClicked
    * This gets called when the tool GUI button is pressed

Properties:

- robotTask

  - mtsDouble toolWeight: weight of the tool
  - mtsDoubleVec toolCom: center of mass of the tool
  - vct6 ftReadingsRaw: raw force torque readings from force sensor
- qtRobotDevice
  - mtsFunctionWrite SetToolGravComp: connects to corresponding function in RobotTask

References

[1] Olds, Kevin. Robotic Assistant Systems for Otolaryngology-Head and Neck Surgery. Diss. 2015.

[2] Dimeas, Fotios Aspragathos, Nikos. Learning optimal variable admittance control for rotational motion in human-robot co-manipulation. In *FAC-PapersOnLine. 48. 10.1016/j.ifacol.2015.12.021.* , 2015.

[3] Murray, Richard M A mathematical introduction to robotic manipulation. *CRC press*, 2017.

[4] Beer, Randall F., et al. "Development and evaluation of a gravity compensated training environment for robotic rehabilitation of post-stroke reaching." *CRC press*, Proc. 2nd IEEE RAS EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob). 2008.

[5] D. A. Fresonke, E. Hernandez and D. Tesar, "Deflection prediction for serial manipulators," *CRC press*, Proceedings. 1988 IEEE International Conference on Robotics and Automation, Philadelphia, PA, USA, 1988, pp. 482-487 vol.1.

[6] Feng, Allen L., et al. "The robotic ENT microsurgery system: A novel robotic platform for microvascular surgery." *CRC press* The Laryngoscope 127.11 (2017): 2495-2500.

[7] Kim, Woo Young Han, Sanghoon Park, Sukho Park, Jong-Oh Ko, Seong Young. Tool Gravity Compensation for Maneuverability Enhancement of Interactive Robot Control for Bone Fracture Reduction System. *CRC press* 2013

[8] Lehmann , Tavakoli, Usmani, and Sloboda “Force-Sensor-Based Estimation of Needle Tip Deflection in Brachytherapy,” *CRC press* Journal of Sensors, vol. 2013, Article ID 263153, 10 pages, 2013.