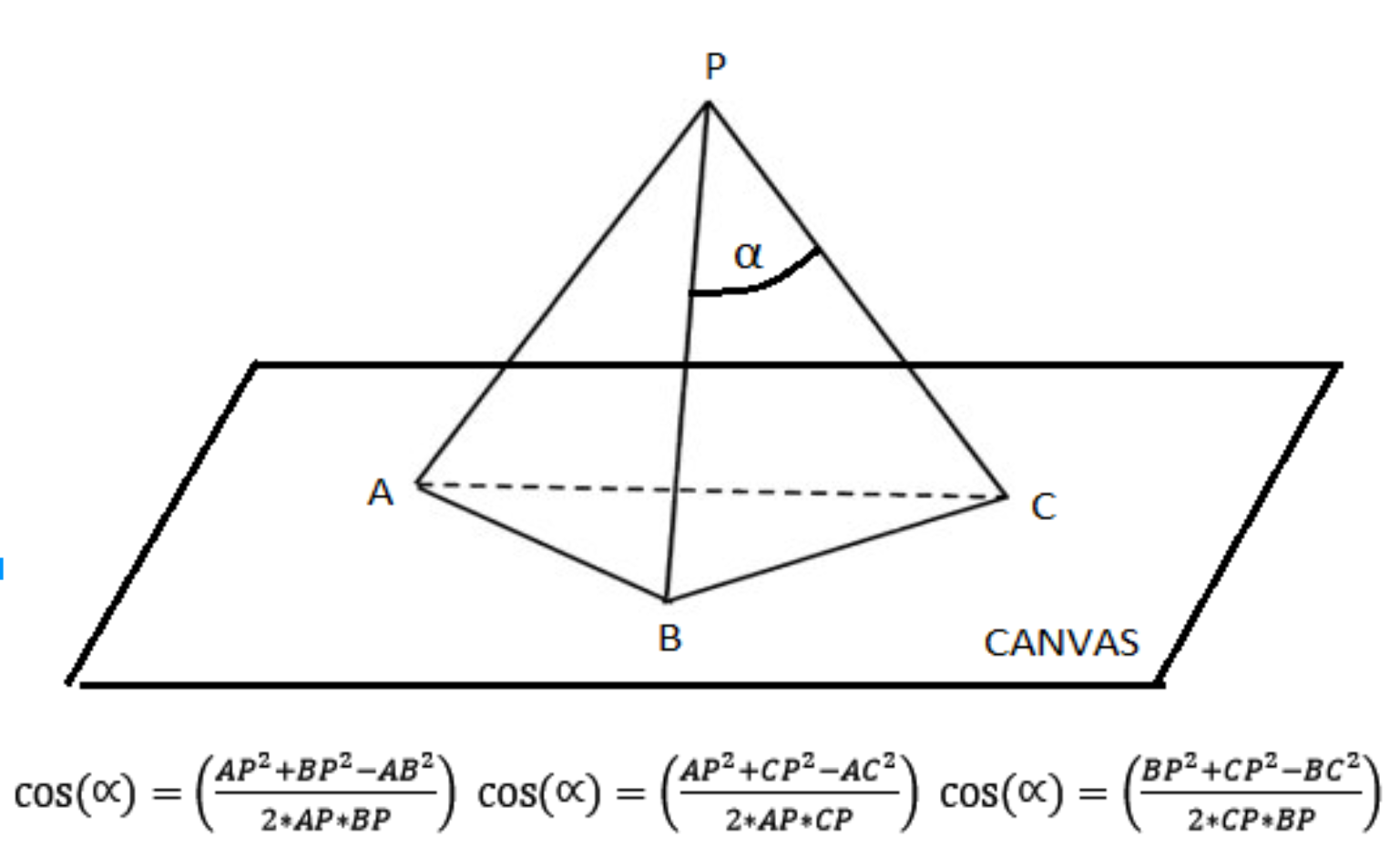
Position Tracking: A 3D Resection Problem

An unconstrained object in 3D motion has six degrees of freedom: three translational and three rotational. Given the projections of three lines on a canvas all which intersect a common source, is it possible to find the position of the source? In finding the solution to this problem, it was uncovered that a similar problem, called the Three-Dimension Resection Problem (3DR) had been solved well in literature. The 3DR problem is to determine the lengths of a tetrahedral given three angles between the sides and the lengths opposite of the angles. The solution, though well documented, is not trivial and with multiple solutions. It is common instead to try to numerically solve the proper solution. Our problem, that of finding the position of the source of three lasers given their intersection with a plane, is somewhat more involved than just solving the 3DR problem. What follows is a breakdown of the procedure including approaches to deal with practical issues that arise.

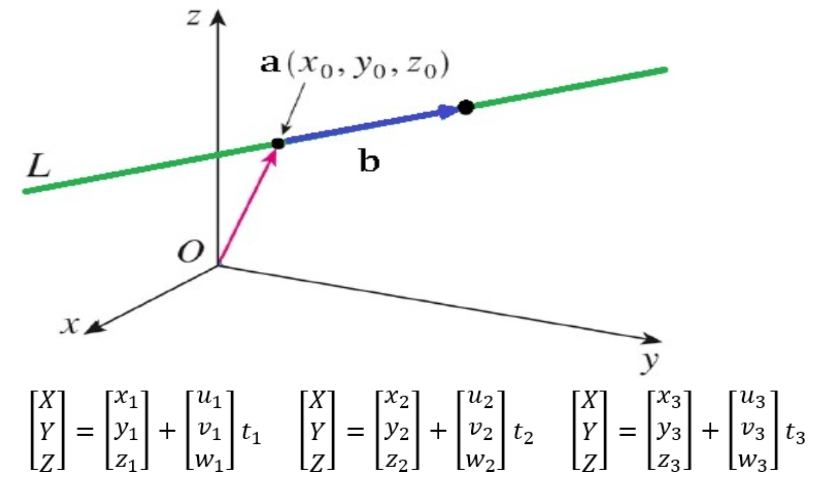
Part 1: Solving for the Lengths

If we examine the geometry that the three lasers make with the canvas, the shape that forms is an irregular tetrahedral. If we consider what we know, the end points of the lasers and the angles between the lasers, the problem of solving for the lengths of the lasers is exactly solving the 3DR problem. Fortunately, the geometry is rather straight forward as the problem boils down to solving the three equations defined by the law of cosines. The issue is this problem has no explicit solution that is trivial, and those explicit solutions which exist, they do not guarantee unique solutions. Here we opt to solve this problem numerically.

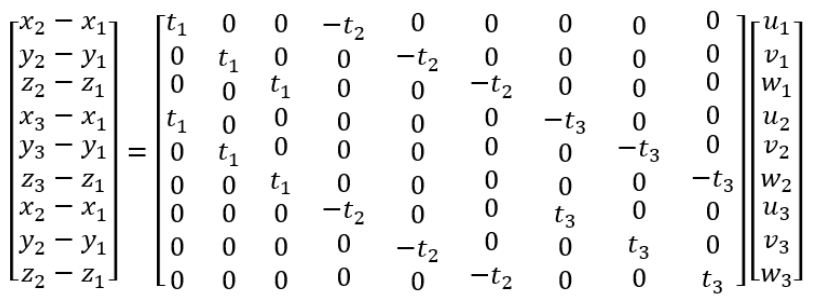


Part 2: Solving for the Direction Vectors

To obtain the source coordinates, we are not just satisfied knowing the lengths of the lasers, but also their directions. With this knowledge we can directly calculate the source position. To solve for the direction vectors knowing the lengths of the lasers, the end points of the lasers, and finally knowing that the lasers have a common source, we can set up three equations vector equations defined by the equation of a line, restricting the lines to pass through one of the three lasers end points A,B,C as well as the common source P.



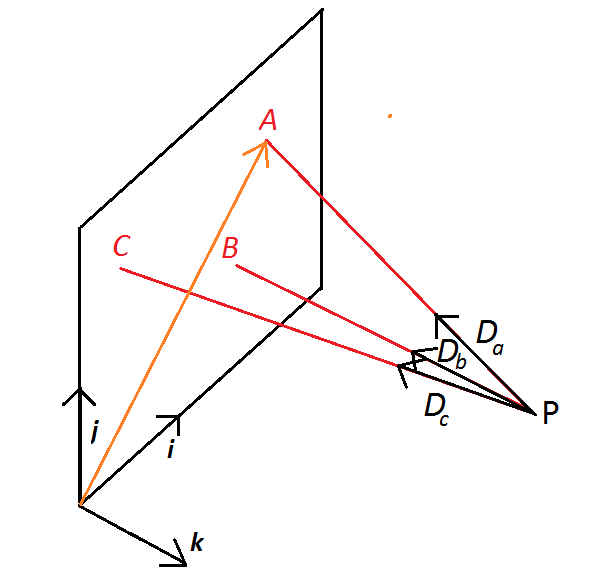
What we obtain are nine algebraic equations with 9 unknowns corresponding to the three entries of the three direction vectors we are interested in. Here represent the lengths of the lasers found, the coordinates of the laser points on the canvas, and finally the unit direction vectors each laser.



At first glance these equations look linear, and they are, however we must restrict the vectors found to be unit vectors, scaled by the previous found lengths. We place this restraint by dividing each vector by its norm. In doing so the system becomes nonlinear, and again a rather nasty set of equations to solve for. How do we then solve for the unit direction entries? Our solution is to solve them numerically.

Part 3: Solving for Position

Now that we have obtained the lengths and direction vectors of the lasers and that we know the coordinates of the laser end points A,B,C we can now obtain the source position simply by vector subtraction.



Part 4: Solving for Orientation (Roll,Pitch,Yaw)

If we consider an arbitrary rotation matrix, there are 9 unknowns. These entries are determined by the angles, axis, and order of the consecutive rotations in sequence. For roll, pitch , yaw the order is a 3-2-1 rotation starting with yaw, pitch, then roll. The rotation matrix has the following form:

Yaw, pitch, and roll angles can be solved by examining elements. It is desirable to define angles in terms of tan2, which keeps track of the quadrant. There are two different solutions for pitch given the expected quadrant it should be in. For our application, we expect pitch to be between -pi/2 and pi/2.

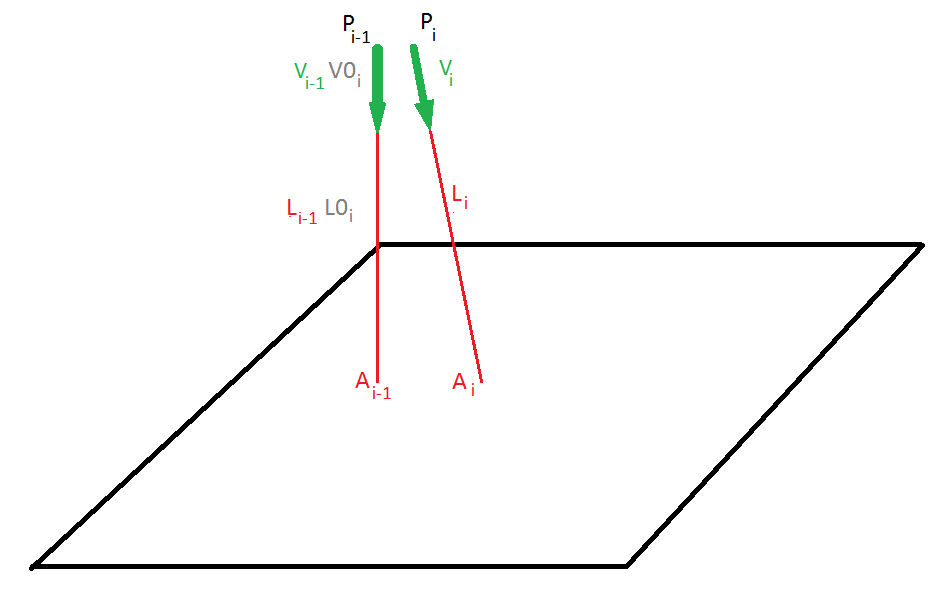
Next, we need to determine how to find the rotation matrix for each update. This is done by considering an initial orientation with the initial direction vectors of the lasers and the current direction vectors of the lasers to determine the rotation matrix which acts to map the two.

Here we have 9 equations in 9 unknowns, as such we can solve this linear system. Here, because of error introduced into the system, we opt to find the nearest solution, in case one does not exist. The system can be rewritten to solve for the components of .

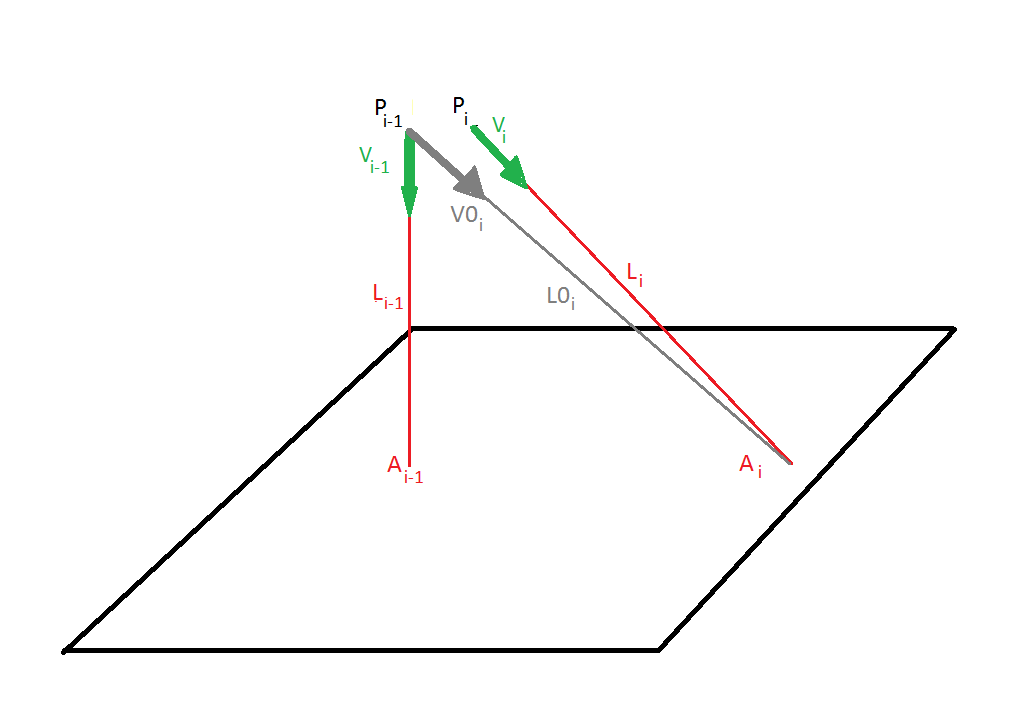
Methods of Implementation

*MATLAB and Python’s fsolve*

In solving the lengths and directions we applied a numeric solver which requires an initial value to process. Unfortunately providing just any initial value will not guarantee the correct value. There are two simple approaches which attempt to avoid the wrong solution. The first approach is to simply use the previous solutions for lengths and directions. This works assuming that the both translation and rotation are slow.



While it may be reasonable to limit the translation, limiting the rotation is simply not realistic. It may be difficult for one to translate a great distance between updates, but it is very feasible to change orientation with the flick of a wrist. For this operation we instead use the previous position estimate and the new known laser end points to estimate the direction and length of the lasers.



Simulations of both cases within MATLAB and Python show that the second approach is more robust to fast orientations changes which are more likely to occur than quick displacements. Both the lengths of the lasers and their direction vectors were solved using MATLAB’s and Pythons fsolve(). Computation time using MATLAB was on the order of .1 seconds compared to Pythons .001 seconds.

*Initialization*

Both solving the lengths and directions of the lasers use a numeric solver which requires the prior position for guessing an initial value. Also the angles between each of the lasers are required. To obtain roll, pitch, and yaw, the initial direction vectors of the lasers are required. All of these values can be found during an initialization procedure upon start up.

Initial Position:

Here we need to know a good guess for position in order to estimate the lengths and directions of the lasers as mentioned earlier. Here we opt to simply ask the user to stand at the left-lower hand corner of the screen corresponding to the origin, and then stand an equal distance from the board compared to the height of the board. For example, if the board is 2m tall, the user would stand 2m away from the lower left hand corner. Because this position is known, this is used to initialize position.

Initial angles:

In order to obtain estimates of our orientation, we must compare the unit direction vectors of the three lasers at a given point in time to an initial, zeroed angle, orientation. There are two ways to do this. The first way is to simply predefine an zero orientation and a corresponding set of laser directions. This is only feasible if you know the angles between the lasers. For example lets assume laser A is pointing in the negative K-axis, laser B is made by rotating laser A by an angle AB about the J-axis, and laser C is defined by the restrictions on angle AC and BC.

The difficulty is in calculating the direction of laser C, because the axis of rotation is not known trivially. The system is constrained, so a relation can be found, however for the purpose of this project, we opted to use the initial direction vectors to define the zeroed angles. This is done by using the initial position and the measured end laser points, and back solving the direction vectors.

Defining angles between lasers:

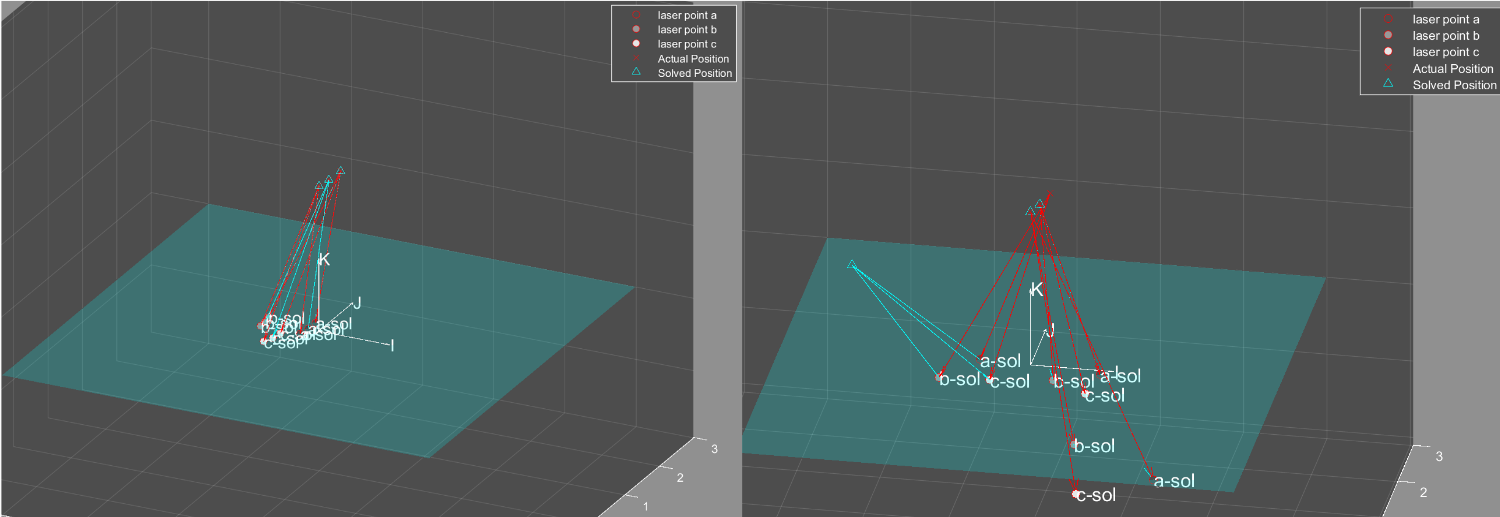
So far we have assumed that we know the angles between the lasers, however if the lasers are not mounted precision, the angles may not be known well. In our case the mounting hardware for the lasers were not rigid as to allow for movement of the lasers to ensure that they intersected a point in space. So every time we tested, the lasers angles may have changed. To get around this issue, we simply back calculated the angles assuming that the initial positon was known, as described above.

In the likely case that the angles found are note equal, the program must be able to track which laser points belong to which laser. In our application we used OpenCV’s contour function, which separates blobs, however it does not keep track of which blobs are which, but instead applies a kernel from the top left to bottom right. The implication of this is that the lasers eventually swap order and the position estimation fails.

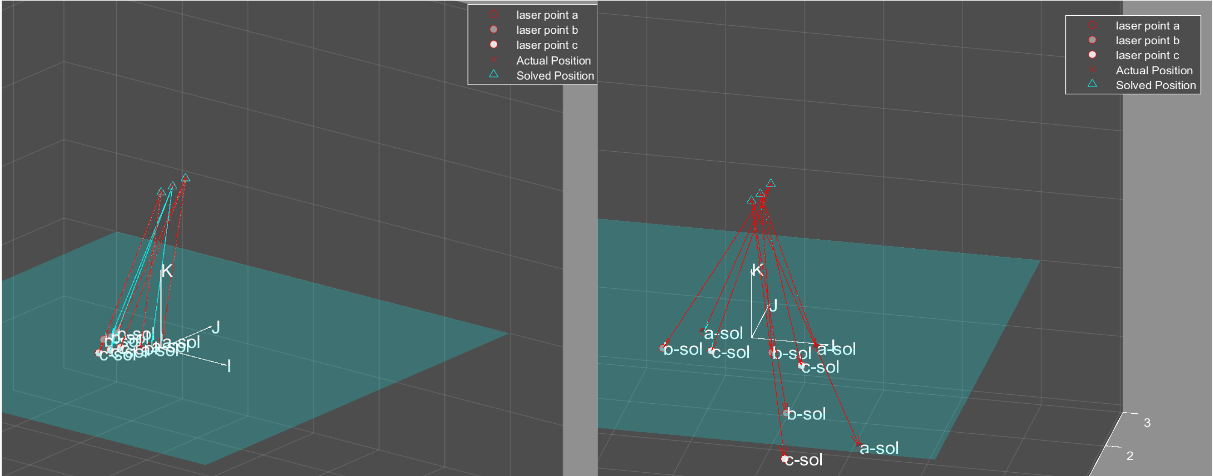
A simple solution to resolve this is to assume that given a correct set of prior laser points tags, that the correct tags for the next iteration will result minimizing the displacements. This was found to resolve the swapping issue.

Simulation:

Two different simulations were carried out, one in MATLAB to visualize the results, and one ported to Python which was the platform used for the project. The MATLAB simulation proved to be a very valuable tool in evaluating the validity of the numeric solutions, though computation time proved to be slow on the order of a tenth of a second. Here it was discovered that using the prior position plus the new laser positions to estimate the new laser lengths and directions proved to be better than just using the past directions and lengths.

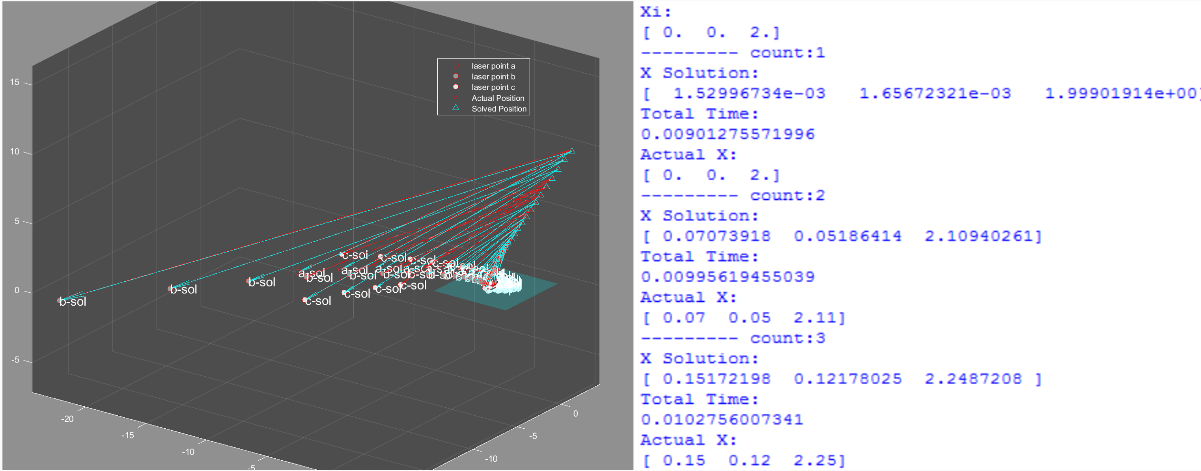


MATLAB Simulation which solves for position by using prior lengths and direction vectors. (Left) Approach is valid for small changes in orientation. (Right) Approach fails for sudden changes in orientation.



MATLAB Simulation which solves for position by using prior position and new lasers endpoints to estimate laser lengths and directions. (Left) Approach is valid for small changes in orientation. (Right) Approach is also valid for sudden changes in orientation.

After the methods passed testing in MATLAB, the program was transferred to Python showing nearly exacting results in much less time. The process operates as fast as 100Hz.



Iterative simulation comparison in MATLAB and Python. The results agree.

Test Results:

The Position Estimation Program runs as one of the demos in the Laser Board Program. To begin the user selects the Laser Position Demo from the main GUI. Next the user is asked to stand at the bottom left corner of the screen at a distance equal to that of the height of the screen, wall while keeping the lasers on the board as close to the bottom left corner as possible. Once the user has done this they can press reset to initialize and run the position and orientation estimation.

The user is given the estimated position in XYZ-coordinates and the orientation in yaw, pitch, and roll angles. Also the user is presented with a circle of varying location and size which is a visualization of their estimated position.

The final results of the tracking demonstrate that position and orientation can be found with reasonable accuracy and speed. The user receives accurate position despite rotation the handles and receives accurate angle despite a change in location. It has been found, however, that estimates can be off if the user does not take care to ensure the initialization is accurate. Also the user must ensure the in fact the lasers meet at a point. If the points do not, then the solution becomes less accurate. That is the geometry of the lasers must make a tetrahedral.