Path Generation by Avoiding Obstacles using the Intersection of Bodies

Komák Martin [1], Králik Marian [1]

[1]*Slovak University of Technology, Faculty of Mechanical Engineering, Bratislava, Slovak Republic*

Abstract **–We will discuss the problem of path finding in 3D space with an obstacle. The thesis deals with the problem of searching for the shortest path between the individual points in the space so that this path does not come into collision with an obstacle. A system has been designed to construct paths in cross-sectional planes of the given object representing an obstacle, based on its surface contour. The system solves the issue of loading STL format, creating cross-sectional planes of the object, intersection between geometric shapes, and generation of lines around the contour of the object in 2D. An experiment was performed, in which we have been moving around a model of a jet aircraft and its results are described in the conclusion.**

Keywords **– path finding, STL format, avoiding obstacles, cross sections, geometric intersection**

1. Introduction

There is currently a number of industries that use the collision-free path finding. This is primarily about finding and plotting the shortest path between two points using a computer application. When dealing with the problem of searching for the shortest path, it primarily refers to the geometric solution resulting from the geometry of the workspace. There are many approaches and methods that deal with planning of the shortest path in an area with obstacles. The best-known algorithms searching for the optimal path in 2D and 3D space are for example: Dijkstra's algorithm, A* algorithm [1,2], Gradient Descent Path Optimizer, Voronoi diagram [3], Visibility graph [4, 5], Probabalistic road map - PRM or Randomly Exploring Randomized Trees – RRT [6, 7]. Many of the methods that are used have arisen in the development of computer games.

An indispensable part of the solution is the appropriate representation of the obstacle in space. One of the most commonly used formats in CAD/CAM systems is the STL format. [8] The STL format consists of blocks of data that define the parameters of the triangles in the space. The surface geometry of the model is built from these triangles. This format is suitable for use in a system for its simple syntax and well-readable geometric properties of triangles. Using the generated triangles, we can simply express the intersection between triangle sides and plane [9]. The individual points of the intersection will define the contour of the object. Based on this expression, we can project the contour of an obstacle into the created cross-section of the object. As a result, the problem of path finding is simplified from 3D to 2D. Generated contour points can then be used to create a path. We use parametric expressions of lines, suitably compiled between each contour point, the start and end points of the path. Appropriate use of the path found can be, for example, creating a path for an industrial robot effector and writing coordinate points into its program.

2. Problem statement

Let us assume that we are moving in the Cartesian coordinate space in which the object representing an obstacle is located. In this space, we want to get from point A to point B. Our goal is to find the shortest path so that it will not come into collision with the obstacle. In this case, the path length is our optimality criterion. The shortest path between two

DOI: 10.18421/TEM71-29
https://dx.doi.org/10.18421/TEM71-29

Corresponding author: Komák Martin,
Slovak University of Technology, Faculty of Mechanical Engineering, Bratislava, Slovak Republic
Email: martin.komak@stuba.sk

Received: 13 December 2017.
Accepted: 07 February 2018.
Published: 23 February 2018.

points is a straight line. If this line passes through an object, we must find another path to circumvent the obstacle and its length will be minimal.

To solve this task, we need to create a system. Each system is based on input values and therefore input data inputs must be created as a necessary part of the program. The whole principle of approach to solving the given task can be seen in Figure 1. The individual parts of the system will be described in more detail in the following chapters.

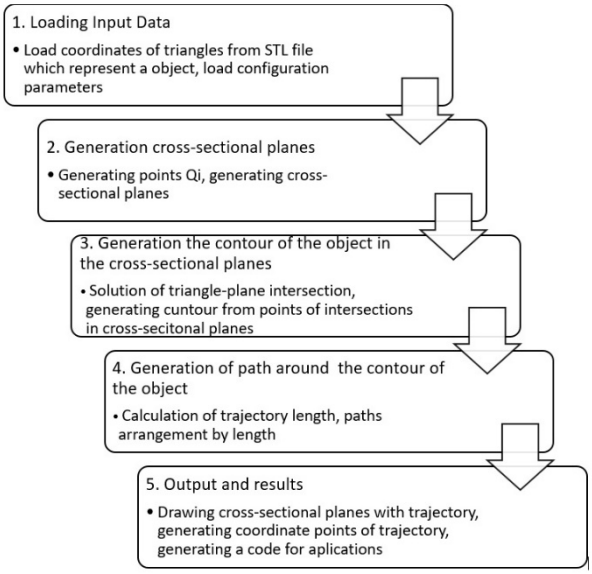

Figure 1. Principle of the solution procedure

2.1. Load STL file

In order to find a path in the space with an obstacle, we have to know, how to define this obstacle in some way. One of the ways to represent any object is by using the STL (STereoLithography) format. The advantage is that data in STL files only describes the surface geometry of a three-dimensional object and does not contain any additional information about color representation, texture, or other attributes common to CAD models. By correct loading this data we get an unstructured surface of the object, consisting of the triangle network described by normal vectors and coordinates of the vertices using the three-dimensional Cartesian coordinate system. This means that the STL object surfaces are approximated to triangles – areas that are bounded by three nonparallel and nonintersecting lines lying in the same plane. In the solution, this triangle system is loaded into memory and is further worked with it. As an obstacle, we chose a simple model of a jet aircraft (Figure 2.).

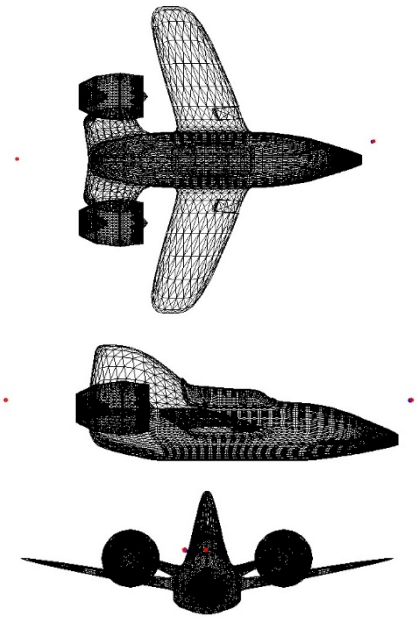

Figure 2. The bypassed aircraft model – STL format

2.2. Load Input data

For additional data necessary for the calculations, the user enters into the system using the created dialog box (Figure 3.).

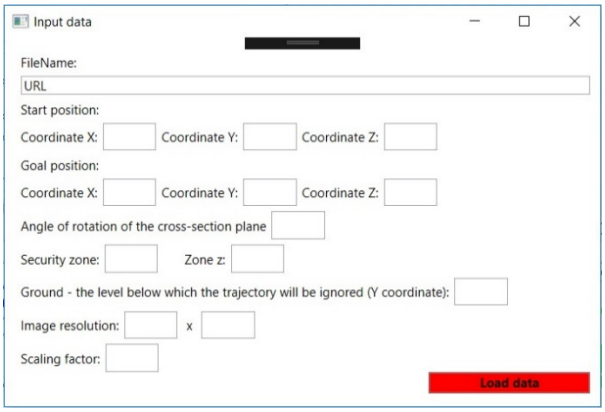

Figure 3. Dialog box for retrieving input data

This dialog box contains a form for entering the following data. In the first row, there is a field to the URL address of the STL file, which represents an obstacle in the space. Next, there are fields for entering the start and end points of the path searched for. Each point is represented by three X, Y and Z coordinates in the 3D space. For applications in industrial robotics, individual path positions are called targets. Let us therefore mark these first two points as $Target_{Start}$ and $Target_{Goal}$. In the next window, we define a safe zone, the value of which determines how far the distance points are offset from the obstacle with respect to their calculated coordinates. The zone z determines the value of approximation we choose for the industrial robot

control program. In the next row of the dialog box, there is an option to choose Ground – coordinate of the vertical axis representing the physical floor. If any point of the generated path has a coordinate under this coordinate, the path is excluded from the result. This setting serves as a baseline representation of the floor, on which the object can be located. At the bottom of the window are entered the height and the width of the output images, showing the generated cross-sectional planes along with the path. The scaling factor is set in the last field, the value of which is set according to the size of the model so that the outline of the model fits into the pictures of the individual cross-sections. From the set values, a configuration file is generated, which the system continues to work with by pressing the Load data button.

3. Algorithm solution

After loading the data into memory, the system begins with calculations. In the first step of the solution, the system connects the linear initial coordinates of the $Target_{Start}$ and $Target_{Goal}$ points. This will give us the shortest path between these points. Now we have to determine if this line is in the intersection with the area of the given obstacle. For this we have used an algorithm which by means of a mathematical expression of geometric shapes, computes the intersection between the line and the triangles representing the object's shell. If the penetration does not occur, this line is found as the shortest path between points. If an intersection occurs with at least one of the triangles, this path is counted as a collision, and the system begins with the calculation of the path that would avoid the obstacle. We try to find the shortest possible path to obstruct the obstacle model without collision. Therefore, we generate a set of paths, which we then compare and evaluate. From the line that represents the shortest path between $Target_{Start}$ and $Target_{Goal}$, we can generate an axis of rotation and call it $Axis_{start\text{-}goal}$. Around this axis we will generate cross-sectional planes $\beta 1$ - βn, with n indicating the number of cross-sectional planes:

$$n = \frac{359°}{\alpha};\ \alpha > 0°$$

where:

α – given angle of rotation of the cross-sectional planes from the configuration file.

This ensures that the cross-sectional planes are evenly formed across all the surfaces of the object. These planes will intersect the object and rotate around the $Axis_{start\text{-}goal}$ axis as shown in Figure 4. The smaller angle α we enter, the denser the cross-sectional planes will be generated, as well as the paths in them. Each cross-sectional plane will contain one path and therefore, by increasing the number of the paths, we will improve the result but extend the computation time.

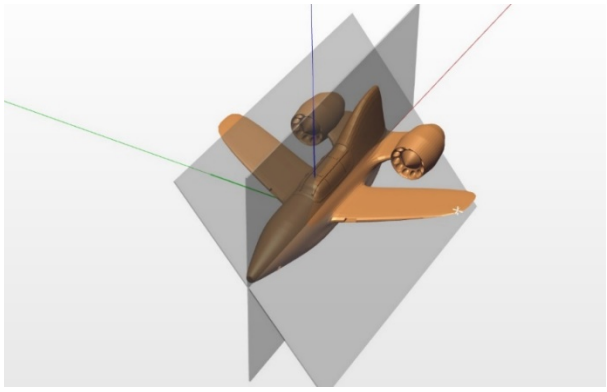

Figure 4. Representation of cross-sectional planes

As a first step in the calculation, we will slightly adjust the space. We will determine the T_{World} translation so that the coordinates of the $Target_{Start}$ point were moved to the origin of the global coordinate system [0,0,0]. Therefore, we determine:

$$T_{\mathrm{World}} = [\,Target_{\mathrm{Start}}X, Target_{\mathrm{Start}}Y, Target_{\mathrm{Start}}Z]$$

We move the $Target_{Start}$ start point by translating T_{World} to the start of the coordinate system. That means, we subtract T_{World} from the original coordinates. The first application is to move the $Target_{Goal}$ point to $Target_{Goal}'$ according to the translation:

$$Target_{\mathrm{Goal}}{}' = Target_{\mathrm{Goal}} - T_{\mathrm{World}}$$

Additional points will already be generated in the modified coordinate system, so they will be moved back to the position according to the real space after the calculations. We make this transfer to facilitate some calculations. Of course, we must not forget that with this translation, it was necessary to also calculate the individual data of the object from the STL file.

4. Generation of cross-sectional planes

In order to define the cross-sectional planes in space, we need to identify 3 points for each of them. The first two points are $Target_{Start}$ and $Target_{Goal}$ located on the $Axis_{start\text{-}goal}$ rotation axis. We need to specify the third point of each cross-sectional plane. The procedure is as follows:

1. We generate the coordinates of any point in the space that is located at $Axis_{start\text{-}goal}$ but is not part of it. We can call it point Q_1.

2. The coordinates of this point along with the angle of rotation α and $Axis_{start\text{-}goal}$ axis are inserted into the matrix of rotation (Chapter 5).
3. By computing, we obtain the points Q_1 through Q_n, located on a center circle in $Axis_{start\text{-}goal}$, as shown in Figure 5.
4. A perpendicular vector stream passing through a generated point to a vector that is identical to the cross-sectional axis obtains the first cross-sectional plane.

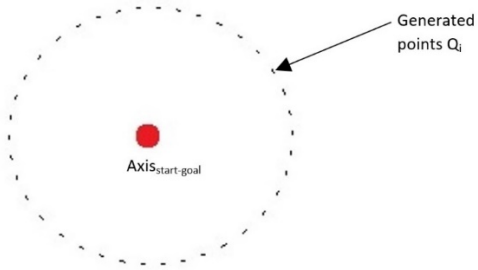

Figure 5. Illustration of generated third - auxiliary points for cross-sectional planes determination

Using the generated third auxiliary points Q_1 - Q_n, we can mathematically express all cross-sectional planes β_1 to β_n. By computing, we also obtain all normal vectors $\vec{N_i}$ for the cross-sectional planes β_i:

$$\vec{N_i} = \overrightarrow{Target_{Start}Target_{Goal}} \; x \; \overrightarrow{Target_{Start}Q_i}$$

where:
i – is the i^{th} cross-sectional plane.

5. Generating object contours in cross-sectional planes

Thanks to the normal line, we can express the individual cross-sectional planes with the general equation of the plane. On account of this expression, we can detect the intersection of the plane and the lines representing the triangle edges from the object's shell. Subsequently, for all the triangles representing the object, we expressed their edges in the parametric expression of a line $p = A + t*\vec{u}$. We have put this expression in the general equation expressing the cross-sectional plane βi. By that we obtain the parameter t_i for point X, in which the line intersects the plane. By computing, we get the intersection between the cross-sectional plane b_1 and the line representing the edge of the k^{th} triangle, with the

triangle k-index from the triangle's net. Subsequently we find the parameters t_1 and t_2 for the end points of the line representing the edge k^{th} triangle. If the t_i parameter of intersection point of the plane is located between the two extreme parameters of the triangle edge line, we can determine the object's contour point in the plane. If this parameter is not between these parameters, the triangle plane is outside the cross-sectional plane, so:

$$t_1 < t_i < t_2 => X \in \beta$$

By successively substituting the parameters into the general cross-sectional plane equation we verify all the edges of all the triangles defining the object and obtain a set of G points in which the cross-sectional plane β_i intersects the object. Then, to draw these points, we need to project these points along with the $Target_{Start}$ and $Target_{Goal}$ points to the 2D plane XY, which will represent the projection on the computer monitor. $Axis_{start\text{-}goal}$ is rotated so that it lies in the X coordinate system axis. For the same angle, we rotate the other points from the set G. The solution can be the use of the matrix of rotation - R, with the angle of rotation being the angle between the vectors $\overrightarrow{Target_{Start}Target_{Goal}}$. Since we have moved the $Target_{Start}$ point to the beginning of the coordinate system, we can use the following relationship to calculate the rotation angle of the axis - δ:

$$\delta = \arccos\left(\frac{\vec{u} \cdot \vec{v}}{\overrightarrow{[u][v]}}\right)$$

where:
$\vec{u}$ is a vector characterizing $Axis_{start\text{-}goal}$ axis,
$\vec{v}$ is a vector characterizing the coordinate X axis.

For rotation, the matrix of rotation R *was used* in this form [10]:

$$R = \begin{bmatrix} \cos б + u_x^2(1-\cos б) & u_x u_y(1-\cos б) - u_z \sin б & u_x u_z(1-\cos б) + u_z \sin б \\ u_y u_x(1-\cos б) + u_z \sin б & \cos б + u_y^2(1-\cos б) & u_y u_z(1-\cos б) + u_z \sin б \\ u_z u_x(1-\cos б) + u_y \sin б & u_z u_y(1-\cos б) + u_x \sin б & \cos б + u_y^2(1-\cos б) \end{bmatrix}$$

After rotating all the points of the set G by the calculated angle δ, we obtain a system that needs to be rotated just around the x-axis. We select any point from the set G and rotate it by an angle γ whose magnitude corresponds to the angular rotation between the vector connecting the selected point with the origin of the coordinate system and the vector representing the coordinates of the coordinate system. Let's use the matrix of rotation again to rotate points from the set G around the x-axis (Figure 6).

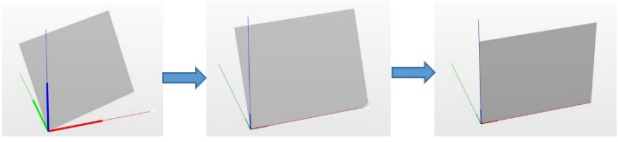

Figure 6. Illustration of the cross-sectional plane rotation in individual steps

By connecting the individual G points, we will create the outline of the component in the given cross-sectional plane β_i. Of course, we connect only the points that are adjacent to one another. The resulting contour can be seen in Figure 7.

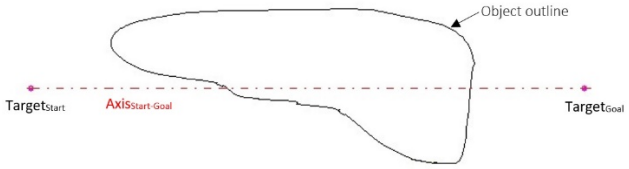

Figure 7. Contour representation of an object in one of the cross-sectional planes

6. Path generation around the contour of a component

As can be seen in Figure 7, we can immediately generate two paths around the contour. It is possible to go around the contour clockwise (top path) and counterclockwise (bottom path). We have determined that we will take the bottom path. Next, we will generate lines passing through the $Target_{Start}$ starting point and the individual points of the G set. That means, we will always generate as many lines, as many points are representing the outline of the component in the given cross-sectional plane β_i. All lines are expressed in a parameter form. We will find out which line has the lowest parameter, i. e. which encloses the smallest angle with the x-axis. From this we get a flowline between $Target_{Start}$ and the selected point P_1. We have generated the first part of the path (Figure 8.).

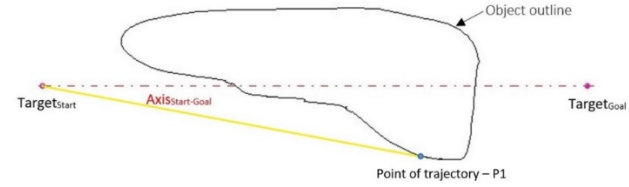

Figure 8. Illustration of first line of the path in the cross-sectional plane

We have obtained the first P_1 transition point, which is also one of the Q_x auxiliary points. The following procedure will be very similar to how the first part of the path was generated. First, we need to generate a subset of G_1. In this subset there will be only the points from the G set whose x coordinate is larger than the coordinates of the obtained point P_1. We also add a $Target_{Goal}$ point to this subset. Then we proceed by generating lines passing through the point P_1 and the points of the G_1 subset. The following procedure is the same as when determining point P_1. We will obtain parameters of lines and determine which point from the subset of G_1 goes through the line with the lowest parameter. We add the point to the G_r set to obtain the resulting transition point coordinates. Again, we start generating lines from this point. In this way, we record the coordinates of the transition points until we reach the $Target_{Goal}$ point. When creating all the lines, it must be ensured that the following lines do not intersect with those that are part of the path.

To calculate the length of the $l_{\beta i}$ path, we must connect the adjacent transition points of the path and count all the distances between them. We calculate the point-to-point distances in the order, depending on how the points were generated. To the sum of these distances, we also add the distances between the first transition point P_1 and the $Target_{Start}$ point and the last P_n point and the $Target_{Goal}$ point:

$$l_{\beta i} = \sum_{j=1}^{n} \sqrt{\left(P_{j+1_x} - P_{j_x}\right)^2 + \left(P_{j+1_y} - P_{j_y}\right)^2 + \left(P_{j+1_z} - P_{j_z}\right)^2} + \sqrt{\left(P_{1_x} - Target_{Start_x}\right)^2 + \left(P_{1_y} - Target_{Start_y}\right)^2 + \left(P_{1_z} - Target_{Start_z}\right)^2} + \sqrt{\left(Target_{Goal_x} - P_{n_x}\right)^2 + \left(Target_{Goal_y} - P_{n_y}\right)^2 + \left(Target_{Goal_z} - P_{n_z}\right)^2}$$

n – the number of transition points,
i– cross-section index,
j – transition point index.

We will gradually generate all the cross-sectional planes and get all paths from them. Subsequently, we sort these paths ascendingly according to their length. Thus, information on the shortest search path will be obtained.

7. Experimental results

We have verified the software solution by an experiment. As an object that represented the obstacle, we chose the model of a jet aircraft - Figure 2. This object was retrieved from the STL file we had available. The file size was 1740 kB. The coordinates of the starting point of the path were (-80, 10, 20) and the (goal) target point (120, 0.1, 20). The angle between the cross-section planes was set to 5°, so 71 cross-sectional planes were generated. From each plane, we obtained one path. All paths were ascendingly arranged by length. We have set the secure zone to zero value, as well as the zone z and the ground level. We set the width of the output images to 1280 and the height to 720. We set the scaling factor to 3.

We have loaded the input data and the entire conversion process run without any error. In Figure 9., we can see a cross-sectional plane with the shortest path. This path contains 5 transition points. The length of this path is 200.533 mm.

Figure 9. The cross-sectional line with the shortest path - $\alpha = 20$ °

Transition point coordinates of the shortest path:

$$\text{Target}_{\text{Start}} = (-80,10,20)$$
$$P_1 = (48.7054,8.28222,21.6915)$$
$$P_2 = (51.0775,8.21159,21.7085)$$
$$P_3 = (55.5932,7.95042,21.6949)$$
$$P_4 = (59.8425,7.56519,21.6313)$$
$$P_5 = (63.8411,7.1019,21.5348)$$
$$\text{Target}_{\text{Goal}} = (120,0.1,20)$$

The resulting transition point coordinates can be written according to a certain syntax into the language understood by the industrial robot control unit. The generated code can be directly used in a real application and visually verify this solution in a selected software (Figure 10).

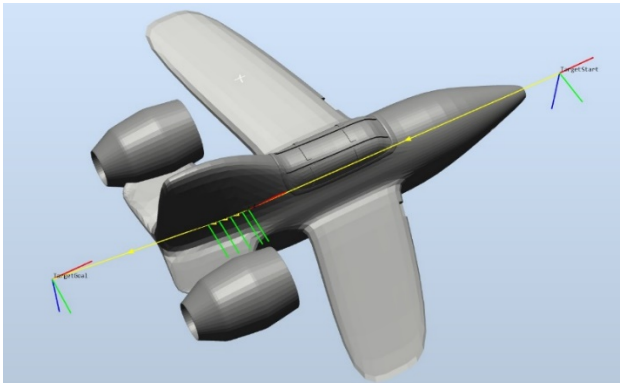

Figure 10. Path visualization in RobotStudio from ABB

Solution generation has low computing and time requirements. The path was generated in a few seconds. Algorithms do not have high computational demands, and the system is not graphically demanding. All results are in the form of *.jpg* images and text files that can be used in the software for off-line programming of industrial robots as well as other applications.

8. Conclusion

The presented results provide a preview of the generated paths around the object in space. Mathematical expressions of algorithms for the generation of paths around the obstacle were presented. The solution was a path without a collision, the transition points of which were part of the cross-sectional plane of the object. Algorithms achieve solutions with minimal computing and time requirements. From a practical point of view, the algorithms mainly work with vector calculations, line equations, and planes. The resulting solution is suitable for many applications searching for the shortest path in an area with obstacles.

Acknowledgment

This article was supported by the Cultural and Educational Agency of the Ministry of Education of the Slovak Republic under the contract KEGA 035STU-4/2017: The introduction of progressive educational methods for manufacturing systems to car production.

References

[1]. Persson, M., & Sharf, I. (2014). Sampling-based A* algorithm for robot path-planning. *The International Journal of Robotics Research*, *33*(13), 1683-1708.
[2]. Pathfinding. (2017, November 12). Retrieved from: https://en.wikipedia.org/wiki/Pathfinding/
[3]. Carbone, G., & Gomez-Bravo, F. (Eds.). (2015). *Motion and operation planning of robotic systems: background and practical approaches* (Vol. 29). Springer.
[4]. Saha, M., & Ante, G. S., & Latombie, J. C., Roughgarden, T. (2006). Planning Tours of Robotic Arms Among Partitioned Goals. *International Journal of Robotics Research*, *25*(3), 207-223.
[5]. Henrich, D., & Wurll, CH., & Worn, H. (2005). Multi-directional search with goal switching for robot path planning. *Computer Science Department, 1416*, 75-84.
[6]. Klingensmith, M. (2016, June 16). Overview of Motion Planning. Retrieved from: http://www.gamasutra.com/ [accessed 18.May 2017].
[7]. Matúšek, O., & Hotař, V., & Laurent, G. J., & Tamadazte, B., & Clevy, C. (2014). Characterization of the Positioning Accuracy and Precision of MEMS Die Servoing Using Model-Based Visual Tracking. *Applied Mechanics and Materials, 613*, 426-433.
[8]. Stereolithography. (2017, October 13). Retrieved from: https://en.wikipedia.org/wiki/Stereolithography/
[9]. Intersection of a Ray/Segment with a Triangle. (2017, November 14). Retrieved from: http://geomalgorithms.com/a06- intersect-2.html/
[10]. Rotation matrix. (2017, November 15). Retrieved from: https://en.wikipedia.org/wiki/Rotation_matrix/