Autonomous Navigation based on 2-Point Correspondence using ROS

Submitted by: Ran Breuer, Li-tal Kupperman
Advisor: Majd Srour, Prof. Ehud Rivlin

Intelligent System Laboratory, CS, Technion

January 26, 2014

Abstract

This project's objective is to implement a ROS node that uses the algorithm and theorems shown in [1]. The project's program implementation addresses the problem of robot navigation using only visual tools in an indoor planar environment. The robot gets as its input a target, in the form of an image taken from the target pose and, using its own builtin camera, navigates itself to the aforementioned target pose. At each step, the robot captures a single image from its current pose and should compute the angles of rotation and distance to the desired target. Under the assumption of planar movement only, this computation is based on a specific and special structure of the Essential Matrix and Homography, allowing the use of only 2-point correspondence when computing - which gives a higher precision and numerical stability. Based on a Pioneer robot with a builtin Axis IP Camera, the robot's control and entire program runs under ROS (Robot Operating System) and consists of several nodes working simultaneously.

INDEX TERMS - VISUAL NAVIGATION, MOBILE ROBOTICS, ESSENTIAL MATRIX, HOMOGRAPHY

1 Background

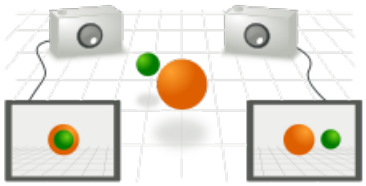


Figure 1: Epipolar Geometry example

1.1 Epipolar Geometry

Epipolar geometry describes a geometric relationship between the positions of corresponding points in two different images.

Given two distinct cameras, each camera has a center of projection denoted O_L and O_R. Both cameras are pointed towards the point P and create two different image planes. The projection of P onto each of the image planes is denoted P_L and P_R respectively.

1.1.1 Epipolar Point

As mentioned above, the two cameras are distinct, and therefore each center of projection O projects onto the other camera's image plane. These two image points, located on the image planes, are the epipolar points, denoted by E_L and E_R.

1.1.2 Epipolar Line

According to the example in figure 2, the line O_L–P is seen by the left camera as a point because it is directly in line with that camera's center of projection. However, the right camera sees this line as a line in its image plane. That line as it is seen by the right camera (E_R–P_R) is called an Epipolar line. Symmetrically, the line O_R–P - seen by the right camera as a point is seen as an epipolar line E_L–P_L by the left camera. An epipolar line is a function of the 3D point P. Since the 3D line O_L–P passes through the center of projection O_L, the corresponding epipolar line in the right image must pass through the epipolar point E_R (and correspondingly for epipolar lines in the left image). This means that all epipolar lines in one image must intersect the epipolar point of that image. In fact, any line which intersects with the epipolar point is an epipolar line since it can be derived from some 3D point X. Given a point in the left image, we don't have to search the whole right image for a corresponding point. This "Epipolar Constraint" reduces the search space to a one-dimensional line.

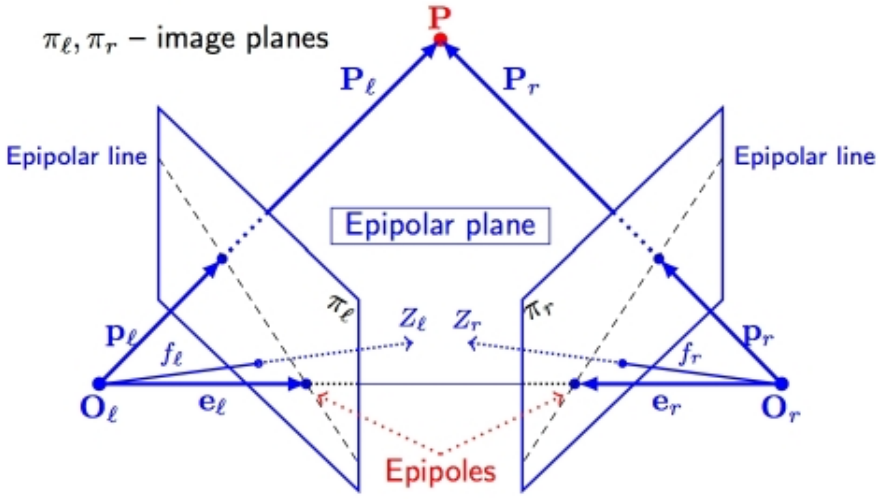


Figure 2: Epipolar Geometry

1.2 Homography

Homography is a 3x3 matrix, which describes the relation between a plane in the world and a perspective image of that plane.

The homography matrix is used for two main purposes:

1. One Image -
   Warping and rotating an image, so it would be correlated with the relevant plane.
   When applying the Homography matrix to every pixel, the new image is a warped version of the original image.

2. Two images -
   Describes how image points from one image are mapped to another image. Meaning, the Homography describes how two images are related to one another. In other words, the homography is used for mapping epipolar lines. It applies only when both images are viewing the same planar scene, but were taken from different positions. Or both images were taken from the same camera, standing in one place, but from a different angle (orientation).

   Using homography, we can write the transformation of points in 3D of the image taken form one of the cameras to the other image, taken from the second camera.

   Given the homography matrix: H and two pixel coordinates: p, p', one for each image, the homography relates the two pixel coordinates, in the two images, as follows: $p' = Hp$ or $p \times Hp' = 0$. The second equation tells us that the vectors p and Hp' are parallel. This is the geometrical meaning of the transformation described above, which transforms a pixel from one plane, to the other, according to the cameras positions.

   In a calibrated case, there is another homography matrix - H_c between the metric coordinates p, p'. The relationship described as follows: $P \times H_c p' = 0$ and $H = KH_cK^{-1}$ (where K is the calibration matrix). Given a known rotation R and calibration K, then the homography H equals KRK^{-1} , and it can be used directly as follows: $p' = KRK^{-1}p$. Applying this homography to one of the images gives an image that we would get if the camera was rotated by R.

1.3 Essential Matrix

Essential Matrix is a 3x3 matrix, which relates corresponding points in two distinct images, of the same planar scene.

Under a rotation matrix R and a translation vector T, the essential matrix is given by $E = [T]_\times R = R\left[R^T T\right]_\times$. The essential matrix is relevant for calibrated cameras, where the inner camera parameters (aka 'Intrinsic Parameters') are known. Under the equation shown above, it holds that $P'^T EP = 0$ where E is the essential matrix, and P, P' are two distinct points in two different image planes. This equation is derived from the epipolar lines. Given a set of corresponding image points, it is possible to estimate an essential matrix which satisfies all the points in the set according to the given images. However, since images are exposed to "noise", which is the common case in any practical situation, it is not possible to find an essential matrix which satisfies all the points in the set exactly. It is possible to determine or estimate an essential matrix which optimally satisfies the given set of corresponding image points. The most common approach is to use the method of solving a total least squares problem. The essential matrix can be useful for determining both the relative position and orientation between the two given cameras and the 3D position of corresponding image points. Once an essential matrix is found, the rotation matrix (and thus the rotation parameters) can be fully recovered and the translation is recovered only up to a scale factor.

1.3.1 Essential Matrix in the Planar Model

In our special case, and the most common, a robot's movement is restricted to the plane parallel to the ground, i.e. the $X \times Z$ plane. The robot's rotation is limited to go around the Y axis only. Hence, the rotation and translation matrices are as follows:

$$R = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \qquad T = \begin{bmatrix} x \\ 0 \\ z \end{bmatrix}$$

We now use polar coordinates for the translation, writing $x = \rho \sin\phi$, $z = \rho \cos\phi$. And so we receive the reduced essential model under the planar movement scene as follows

$$E = \rho \begin{bmatrix} 0 & \cos(\theta - \phi) & 0 \\ -\cos\phi & 0 & \sin\phi \\ 0 & \sin(\theta - \phi) & 0 \end{bmatrix}$$

2 Algorithm(s) used

2.1 Introduction

In this section we will review the various parts and phases that construct the navigation algorithm. We wish to move a robot to an unknown target position and orientation denoted by S, which is specified by an image Itaken from that position. Denote the current unknown robot's position by S', our goal is to lead the robot to S. We assume the camera is calibrated and that its intrinsic parameters are already known. To determine the motion needed by the robot we need to recover the position and orientation of the robot relative to S. given that I and I', the images taken from S and S' respectively, have a large enough overlap, we can extract matching feature correspondences from these images and estimate an essential matrix. By finding the essential matrix E that satisfies as many correspondences as possible, we can extract the rotation and translation matrices from E and from those determine the motion needed to get the robot to S. While the rotation matrix and parameters that construct it can be fully recovered, the translation matrix can only be recovered up to a scale factor, forcing us to use another image to accurately calculate the distance to S.

2.2 Program Flow

The navigation process uses the following algorithm scheme:

1. Given a target image, capture an image from the current pose.
2. Extract and match features from both images
3. From all matches, filter the largest inliers set using some variation of PROSAC.
4. Using the best model estimated, extract matrices R and T.
5. Calculate the distance to target, explained below.
6. Move to a new pose and repeat 1 until target pose is reached

This algorithm uses an estimated essential matrix to determine the robot's next move, therefore there are several iterations required to get to the desired target pose at a certain precision.

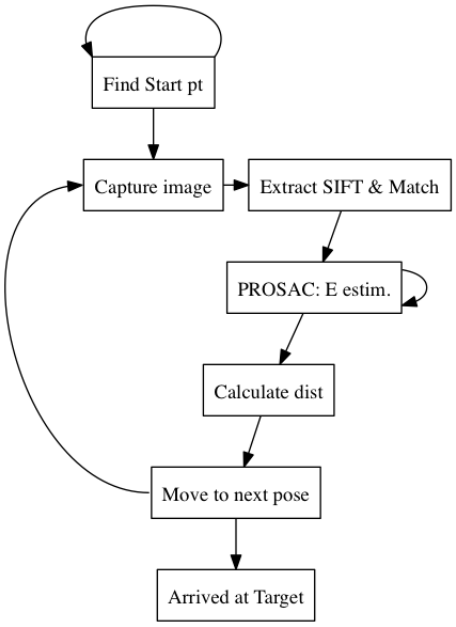


Figure 3: Program Flow Diagram

2.3 Finding a good starting point

Before we can start following the algorithm mentioned above, given we cannot know the robot's location relative to the target, and not even be sure that both images I and I' will have large enough overlap so that matching features will be found, a good starting point for the algorithm needs to be found. We search for a good starting point, i.e. a pose where an image that would be taken from will have large enough overlap with the target image. Starting at a certain position, we would like to scan the space around the robot in an efficient enough matter so that we can cover the whole room and take a 360° images of the room. As soon as we find a good enough spot to start the algorithm, we stop the search cycle and go to step 1. The search algorithm scans the room in an almost circular motion, where on each stop the robot scans on several angles.

2.4 Finding Correspondence

In order to find right correspondences, visual features are extracted and matched from both images. Feature extraction is done by using the SIFT method, as it is scale invariant, matching is done by *OpenCV*'s Flann feature matcher (Brute Force Matcher can be used also). Out of the features and matches formed, we filter those whose distance ratio is higher than a certain degree, this is used to eliminate "noise" or wrong and unnecessary samples from which to estimate the model. After having extracted and matched features from both images, we try to find the largest subset of inliers that will agree on the same model, i.e. E, the essential matrix estimation. We use a variation of the PROSAC (**PRO**gressive **SA**mple **C**onsensus) method paradigm to find the largest set of inliers from which we can extract a better estimation for E.

2.5 Estimating the Essential Matrix

As shown in (1.3.1) in our case, the essential matrix is reduced to the form of

$$E = \rho \begin{bmatrix} 0 & \cos(\theta - \phi) & 0 \\ -\cos\phi & 0 & \sin\phi \\ 0 & \sin(\theta - \phi) & 0 \end{bmatrix}$$

where θ is the rotation angle around the Y-axis, and ϕ is the angle needed to take to get to the desired target pose, where its translation in polar coordinates is $x = \rho \sin\phi$, $z = \rho\cos\phi$.

Due to the number of degrees of freedom (DoF) - 2 - we need only a 2 point correspondence to estimate an essential matrix. However, as we don't know in advance which 2 points to choose in order to get the better and more accurate result, we use our PROSAC iterations to get the matrix E with which the largest set of points (inliers) agree. As shown in [1], given points in I and their match in I', $p_i = (x_i, y_i), \tilde{p}_i = (\tilde{x}_i, \tilde{y}_i)$ respectively, we can use the fact that for each i: $\tilde{p}_i^T E p_i = 0$ and assemble these equations in the form of a linear equation matrix (1) $Ax = 0$ where

$$A_i = \begin{bmatrix} \tilde{y}_i & -x_i\tilde{y}_i & y_i & \tilde{x}_i y_i \end{bmatrix} \qquad and \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \quad (2)$$

where $E = \begin{bmatrix} 0 & x_4 & 0 \\ -x_2 & 0 & x_1 \\ 0 & x_3 & 0 \end{bmatrix}$. However, as we only need 2 degrees of freedom, we can modify these equations so that we can use only two equations to solve this linear equation system. We modify the problem to an optimization problem:

$$argmin\left\{\|Ax\| \quad A \in \mathbb{R}^{2\times 4}, x \in \mathbb{R}^4\right\} \quad (3)$$
$$s.t \quad x_1^2 + x_2^2 = 1 \ \wedge \ x_3^2 + x_4^2 = 1$$

As proved in [1], this problem is similar to finding the intersection between a unit-circle around the origin and an ellipse formed by the diagonal matrix from the SVD extraction of the aforementioned optimization problem. When finding the intersection there can be:

1. 2 (or 4) intersection points - our essential matrix solution derives from that.
2. no intersection points - meaning there is no solution found from the 2 point correspondence.
3. an infinite number of points - meaning this is a Pure Rotation scenario, which will be discussed later.

Once we found a good essential matrix estimation, we now try to find, using PROSAC runs, the solution that holds (1) for as many matches as possible, i.e. the largest support set of inliers.

From our essential matrix solution we have just computed, knowing that it is supported by the largest set of inliers possible, we can extract ϕ, θ using them to determine the direction in which to direct the robot, and its rotation (or orientation modification) to be made.

2.6 Pure Rotation

A pure rotation scene describes a scene where the robot's position is at the target, however there is a difference in orientation. This can be resolved by a 1 point correspondence between the target image and source image. If we

are at a pure rotation scene, the essential matrix will consist of only the influence of θ and thus require only one equation to solve. However, with using only an estimation for the essential matrix, we find a probability for a scene to have pure rotation traits, where if the probability is high enough we deduce it is a pure rotation. Note that in our implementation, we recognize that we are at the target location and orientation by the fact that we are in a pure rotation scene constructed by a low angle θ.

2.7 Calculating Distance

After Calculating the rotation angle θ and the direction to the target pose, ϕ, there is still the matter of the distance ρ which is needed to advance to the target. In [2, 3] distance is determined by moving toward the target in the direction determined by ϕ and by a constant step λ. The amount of steps needed to advance to the target is calculated by the factor of improvement after making the initial step. In addition, in our implementation, a different approach is used as well. To determine the distance to the target, we move the robot a predetermined step and direction to another pose S", and take a 3rd image I". By using the same algorithm for essential matrix extraction from I", we get the angles θ' and ϕ'. Now, we use the 3 poses S, S', S" that form a triangle whose angles we can recover using our knowledge of θ and ϕ from both previous locations. Using basic trigonometry and the fact that we know one side of this triangle (the step we took from S' to S") we can now recover the exact distance to the target. In our implementation, we implemented both methods of distance computation and compared them both.

While the triangle method has been proven to be more accurate in common cases, the Cross-ratio (a.k.a 'X-ratio') method, as seen in [1] is less prone to errors in certain situations. The triangle method suffers from large errors due to "flat triangle" situation, in which the 3rd pose's image is of insignificant difference from its predecessor, resulting in small deltas between computed and angles and thus numeric cancelation occurs.

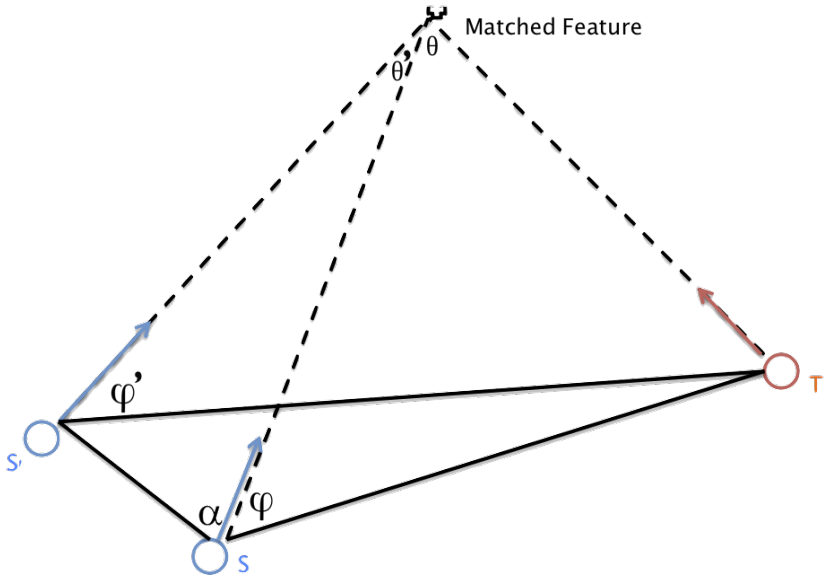


Figure 4: Calculating distance to target by 2 poses

3 Design and Implementation

Our implementation of this algorithm is built on a Pioneer robot unit with an Axis IP Camera built on it. Using several ROS nodes to communicate with the different parts of the algorithm, we implemented a ROS node that runs the algorithm on the targets it is given.

3.1 Design Concepts & Modules

The implementation is constructed of several modules, where each performs a service for the main module, the Auto Navigator. The Auto Navigator is given a target in the form of an image file, indicating the next target pose to go to by using the algorithm mentioned above. Then, by using the Robot Controller and Image Capturer to move the robot and get the images from the camera at its pose. The data, both target image and the newly captured image are then transferred to the Epipolar Solver, where the angles θ and ϕ are extracted. Then we use these angles as the next step towards the target. This whole process is repeated until we reach a pure rotation scene or the number of iteration exceeds the maximum set as a limit.

3.1.1 Auto Navigator

The Auto Navigator is our node's main module, it holds instances of all other service modules, combining them all in order to navigate the robot efficiently through the algorithm.

Each instance of this class can navigate automatically to a designated target pose in the form a target image, or a path to an image file containing this target.

```
class AutoNavigator {
        RobotController controller;
        ImageCapturer imgCapturer;
        ImageUndistorter undistorter;
        EpipolarSolver epiSolver;
        ...
public:
        AutoNavigator(int argc, char* argv[]);

        AutoNavigatorResult goToTarget(const char* targetFilename,int maxIter = 50);
};
```

Auto Navigator: Interface

The Auto Navigator can navigate to any given single target, if number of iteration exceeds the limit given or starting point isn't found - a failure message is returned.

3.1.2 Robot Controller

In order to move and rotate the robot to where we desire, we will use the Robot Controller interface. The controller enables a simple abstraction of robot maneuvering and movement control, using simple $Move$, $Rotate\&TwistAndMove$ commands. This module communicates with the ROSARIA node (explained below) and transfers commands via /cmd_vel/Twist messages.

```
class RobotController {
        ...
public:
        RobotController(int argc, char* argv[]); //Ctor

        /* order of execution is rotate first - move second.      */
        void twistAndMove(float angle, float distance);

        void rotate(float angle);
        void move(float distance);
};
```

Robot Controller: Interface

3.1.3 Image Capturer & Undistorter

Our robot is equipped with an Axis 207MW IP Camera, which when turned on broadcasts a stream of images at 30 fps. As our implementation only requires one image (frame) per iteration, we use the Image Capturer to get a single frame from the camera, by simply calling the captureImage() method. However, upon image retrival from the camera, it is distorted, parallel line may seen crooked, and the depth of field is deceiving. We now need to call the Image Undistorter which, as its name suggests, undistorts the image given by the camera using the calibration matrix and distortion coefficients recovered when we calibrated the camera.

Figure 5: An original distorted image from the camera & its undistorted unidentical twin

```
class ImageCapturer {
        ImageUndistorter undistorter;
public:
        ImageCapturer(int argc, char* argv[]);
        ~ImageCapturer();
        Mat captureImage();
};
```

Image Capturer: Interface

```
class ImageUndistorter {
public:
        ImageUndistorter();
        ~ImageUndistorter();
        void Undistort(const cv::Mat& src, cv::Mat& dst);
};
```

Image Undistorter: Interface

3.1.4 Epipolar Solver

The Epipolar Solver is the heart of the navigation process. It encapsulates the essential matrix approximation, SIFT feature extraction, feature matching process and many more within. The Epipolar Solver also enables distance computing given the results from both images processing and angles extraction. All image manipulation and processing is done by using *OpenCV*'s functions in $C++$.

Feature extraction is done with the SIFT feature extractor and feature matching is done with Flann Matcher. Distance recovery is done by retrieving all of the aforementioned triangle angles and computing the missing side length (2.7).

3.1.4.1 PROSAC Essential matrix Estimation The essential matrix estimation is based on the following algorithm:

Algorithm 1 PROSAC Essential Matrix Estimation

1. Extract SIFT features from both images
2. Match features and filter 'bad' matches
3. FOREACH 2 matches:
   (a) Create an essential model E based on the 2 matches as shown in (2.5)
   (b) Collect the largest subset S_E of inliers that agree with E
4. return the essential model E where S_E is the largest.

```
class EpipolarSolver {
public:
        EpipolarSolver();
        virtual ~EpipolarSolver();
        class notEnoughPoints : public exception {};
        class noEssentialSolution : public exception {};
        ...

        int findSiftFeatures(
                        Mat& image1, Mat& image2,
                        vector<Point3d>& MatchedOutput1,
                        vector<Point3d>& MatchedOutput2 ,
                        vector<double>& distanceRatioVec);

        int prosacEssential(const vector<Point3d>& inputPoints1,
                                                const vector<Point3d>& inputPoints2,
                                                Mat& outputMat,
                                                vector<int>& bestInliers,
                                                float threshold,
                                                int maxTrials, int maxNoImprove);

        int calculateAnglesFromImages(Mat& img1, Mat& img2, double& theta,
                                                double& phi,int& dir_sign);

        double calcDistanceToTargetTriangle(double alpha, double theta1, double phi1,
                                        double theta2, double phi2, double x);

        double calcDistanceToTargetXRatio(const vector<Point3d>& framesFMatchedFT, const vect
                                        const vector<Point3d>& framesFMatchedFS, const vector
private:
        ...
}
```

Epipolar Solver: Interface

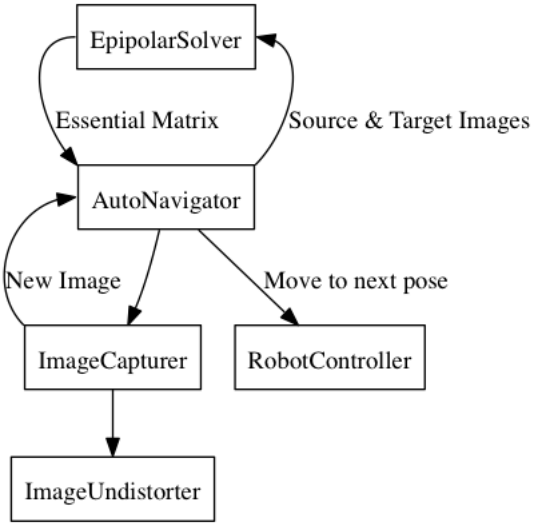


Figure 6: Auto Navigator Interface Diagram

3.2 Using ROS

ROS is an open-source, meta-operating system used on robots. It provides services similar to other operating systems, including hardware abstraction, low-level device control, message passing between processes and many more. ROS works with *nodes* - a ROS Node is a process that runs independently and performs a certain task. A node can **subscribe** or **publish** to a *topic,* this is the way to communicate between nodes. Each topic is considered a **service** to which a node (task) can publish certain data and\or another node can retreive data from.

Working with ROS enables a simple abstraction when working with robots, as it provides easy to handle wrappers for mechanical operations, e.g. moving the robot somewhere or taking a picture from a webcam.

In our project we have used the following packages and nodes:

3.2.1 ROSARIA

ROSARIA is a ROS package that allows simple manipulation of various robots through a very simple API. It is a ROS wrapper for MobileRobots' ARIA C++ library. ROSARIA allows controling a robot by simply setting its velocity (angular and linear) and retrieving its odometry data too.

The package subscribes to the **cmd_vel** topic and through it receives new velocity data in the form of a **geometry_msgs/Twist** message (including angular and linear velocities), and thus setting the new parameters to its assigned robot, in our case the Pioneer robot.

3.2.2 Axis Camera

The Axis camera package subscribes to a webcam attached to the computer and publishes its **photo stream** to a specified topic in the form of a **image_raw/compressed** message. With only subscribing to the aforementioned topic, our program, in our Image Capturer module, can receive the photo stream and use it to 'capture' a single frame.

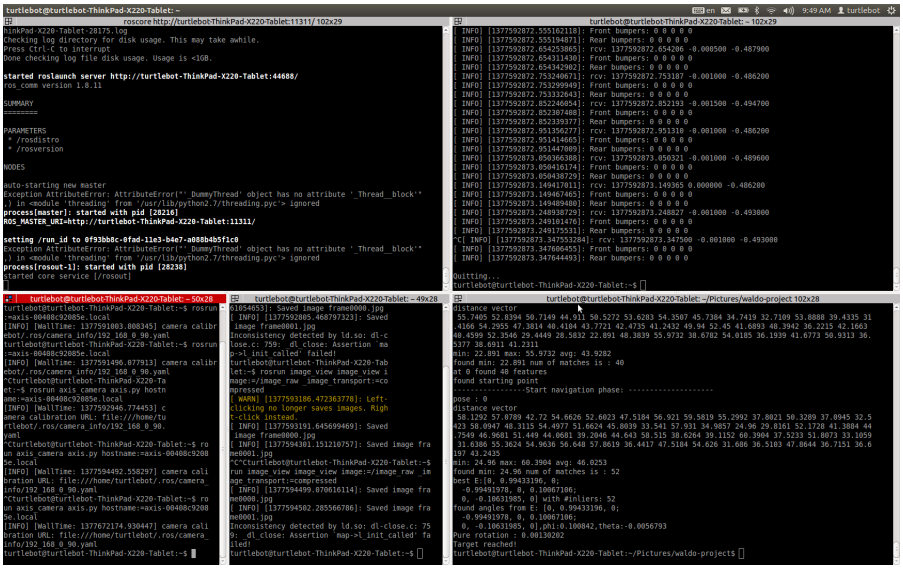


Figure 7: ROS Nodes working simultaneously

4 Challenges

4.1 Eliminating bad images

When collecting the source images from the current pose, there is a matching process based on similarity between image features. The Flann Feature Matcher and the Brute Force Matcher OpenCV provide, along with every other feature (& descriptor) matchers, construct sets of pairs of matched features up to a certain thershold. However, when matching, some of the matches can be inaccurate due to the nature of the matching (matching specific features and not looking at whole (bigger) picture) resulting in rough matching that sometimes can match completely different scenes and call them identical. When searching for a good starting point (2.3) and further on when computing the data needed for navigation (2.4) we need to be able to rely on the matching being true to the scenes. By setting a minimal thershold parameter for match distance, we are able to verify that the matching will only get matches close enough to provide accurate results.

The threshold is set based on the minimal distance of all matches found between the two scenes - t , and we set the threshold to be λt where λ is a factor set in advance. With this thershold set we assure only the 'good' matches will help computation. More over, after setting a max for this threshold, we eliminate bad scenes from being even considered for matching. This being a result of the fact that even a scene that has absolutely no relation to the target scene & pose, has a match with a certain distance (which would be much greater than our max threshold).

In short, by assuring that a set of matches S, where the best (with closest distance) match $m*$ to the target scene is under a threshold t_{max} and every other match holds $dist(m) \leq dist(m*) * \lambda$ - we are sure to eliminate all bad images from interrupting the computation process and thus the whole navigation procedure.

In the tests ran, we set $t_{max} = 90$, $\lambda = 2$.

4.2 Algorithm divergence near target

The algorithm provided above is found to converge to the target up to a certain point. When reached that point, where distance to target T is close and we reach a close to 'Pure Rotation' scene - the algorithm starts diverging and moving the robot away from the target.

A solution to this problem is setting the target to be any pose with orientation θ and distance d relative to the target so that $\theta < \theta_{max}$, $d < d_{max}$.

In case of a small difference between two given images, defined by the maximum absolute distance between matching features, if said maximum is under a predefined bar - an exception will be thrown to indicate we are close enough to the target.

In our tests ran we set $\theta_{max} = 5°$, $d_{max} = 0.1_m$.

4.3 Computing distance to target - Triangle method

As shown in 2.7, the distance to the target pose, in the triangle method, is computed by using two poses S and S' and extracting φ, θ from both poses. By using basic trigonometry and triangles realtions to extract the distance from S' to the target. However, if the target T is too far from S and S' the difference in angles extracted between the two poses is too small or insignificant to compute the distance well. When receiving too small of a difference between φ, θ , under a minimum set value ϵ, there is a large chance of Numerical Cancelation between values and thus receiving wrong (and many times very large and inaccurate) results.

To reduce the chance for numerical cancelation, we modify the calculation of the distance so that the number of subtractions is minimal. However, we still have a problem where the two poses are closer to each other than to the target.

To solve this issue, we check the difference between the two poses (by the angles extracted from them), if the difference is under a certain value - we deduce this is an error-bound computation and issue a small result to just advance the robot towards the target (in the direction φ).

4.4 Restarting after bad step

While computing the next step and estimating the model is proven to converge, sometimes a bad step (or an auxiliary step to compute distance) can cause the robot to lose an angle or view of the target scene or pose. We introduced a 'Back step' procedure to the robot: As soon as the robot notices it is out of his way, he "back steps" - takes a random short step to get out of the current pose and starts looking for a new starting pose (2.3).

Bad position recognition is done by the robot simply not recognizing any matches over a 'good' level, contradicting the assumption where all poses along the route taken to the target.

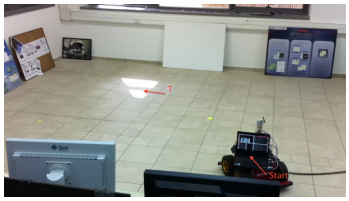
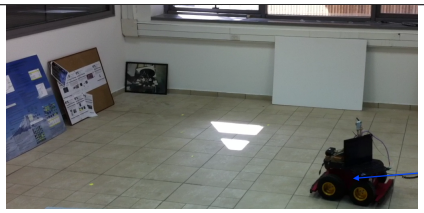
By enabling the robot to reset its algorithm run - we enable the algorithm to run smoothly and not 'get stuck' on a fault.

5 Experimental Test Results

Tests of the implementation were ran on:

- Pioneer Robot (complete model here)
- Axis Web camera
- Laptop running ROS as master node.

When running the implementation from any random position in a room, the algorithm converges the robot in 3-5 iterations to $\sim 40_{cm}$ from the target pose with 'Triangle' distance approximation method. As written above, from this point on - the algorithm is diverging, resulting in distancing the robot from the target. When using the 'X-ratio' distance method, convergence seems to be slower but more efficient in any distance to target, resulting in a much better end result.

		
Starting position	1st iteartion	2nd iteration
		
3rd	4th	End - 6 iterations (distance: $\sim 20_{cm}$)

Above is a sample run of the algorithm using 'Triangle' method. Below is the target image taken from the target pose, and the image taken from the reached pose at end of the algorithm.

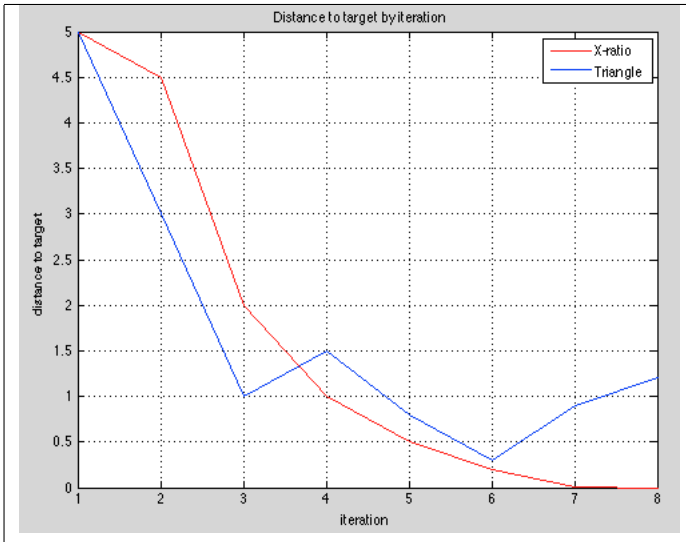
	
Starting position	Midway through run - after 2 iterations

Target Image

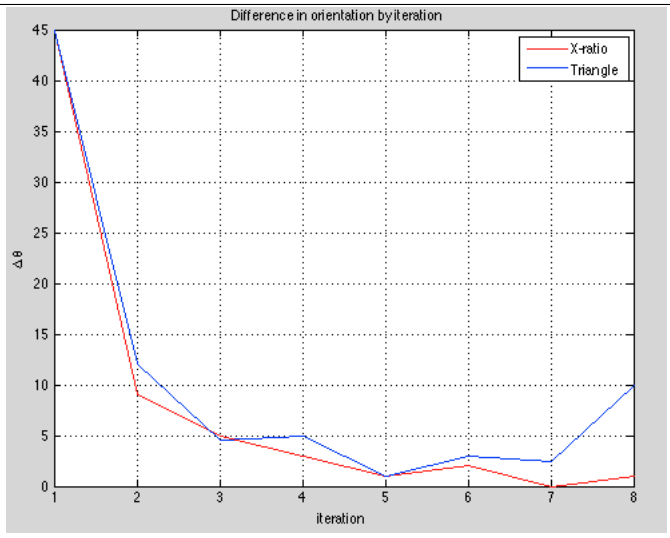
End of run captured image

5.1 Measurables

Below are the convergence of the distance to target and difference in orientation through the algorithm's progress, by iterations.



Distance to target by iterations



$\Delta\theta$ - Difference in orientation

By comparing both distance approximation algorithms, we can see the quicker convergence of 'Triangle' method when we are far from the target. However, the 'X-ratio' method is found more useful in any distance for it is producing a better and closer end point data. While the 'Triangle' method is more accurate when given a big enough difference between poses, the 'X-ratio' is independent of pose distance to target and giving the same results wherever it may be.

Also, when close enough to the target pose, 'Triangle' method diverges as mentioned before, which results can

be seen upon distancing itself from the target at later stages of the run.

6 Summary, Open issues and Future work

6.1 Issue: Algorithm failure under a planar scene image

When using the algorithm for Essential matrix approximation, we sometime come across a dominant plane scene. A dominant plane scene is one where the majority of the image, or more precisely - Image features, reside on the same plane. These scene usually include a wall (or some), picture etc. In our algorithm, when extracting the image features and approximating the essential matrix through them, when coming across a dominant plane scene - most features (and thus inliers of the same model) are on the same plane (e.g. wall) resulting in 2 essential matrices, both of which are solutions.

Usually, when this occurs, we will receive 2 possible solutions while one has a significant amount of inliers while the other doesn't. To resolve this issue, another computation should be made, as mentioned in [1] briefly. The solution is to take another image from a random pose facing the same scene. While we still receive 2 solutions for E, the solution regarding the dominant plane will change in accordance to the plane's normal vector in the new image. The other solution, the one not influenced by the dominant plane and its normal's variations, constructs our "true" essential matrix solution.

In our implementation, we do not overcome this issue, nor implement the aforementioned method to resolve it. In our test runs we give as much of a non-dominant plane scene as a target, to prevent miscalculations due to this matter.

6.2 Summary and Future Work

As shown in this project, and the previous work mentioned, Vision based navigation is able to deduce orientation and distance to a given target by a number of computations using **Feature matching** and **Essential Matrix** approximation and extraction. Once an Essential model is acquired, rotation and translation matrices are easily recovered and the next robot's movement is known.

In this project, we implement a model based on previous knowledge of the scene. While the images given can be different and almost non related, we can always assume **Planar movement** and **Y-axis only rotation**, giving us a better idea of the essential matrix structure, thus having a better and more efficient way of computing the rotation and translation matrices.

The algorithm shown above constructs an essential matrix approximation using only **2-point correspondence** between two images, a **Target image** & and a **Current position image**. By using LS approximation and a linear equation solution to the problem we can compute the model needed for the navigation process.

By implementing the algorithm and testing its runs, we can see that the algorithm quickly converges to a close enviroment of the target pose and orientation (T, θ). However, It is found that in close enviroments, close to the target pose, the computation starts diverging again away from the target.

In future work we recommend:

1. Solving divergence at target pose.
2. Better elimination of 'bad' image sources (wrong poses).
3. Qualifier for better matches and features for a more efficient estimation each step.
4. Better and\or more efficient method of distance computing

References

[1] Ehud Rivlin Boris Cherevatsky, Ilan Shimshoni. Autonomous navigation within an indoor environment. *Technion*.

[2] Ilan Shimshoni Evgeny Smolyar, Ehud Rivlin. Image-based robot navigation in unknown indoor environment. 2003.

[3] I. Shimshoni R. Basri, E. Rivlin. Visual homing: Surfing on the epipoles. *International Journal of Computer Vision*, 1999.