Department of Electrical Engineering,
Syed Babar Ali School of Science and Engineering,
Lahore University of Management Sciences, Lahore, Pakistan

VISUAL SERVOING OF A ROBOTIC MINE DETECTOR ARM

By

Ali Musa Iftikhar	**Roll No: 14100044**
Hamza Anwar	**Roll No: 14100048**
Muhammad Furqan Afzal	**Roll No: 14100182**
Qasim Zafar	**Roll No: 14100086**

A report submitted in partial fulfillment of the
requirements for the degree of

BS Electrical Engineering

Syed Babar Ali School of Science and
Engineering,

Lahore University of Management Sciences

Under the supervision of

Dr. Abubakr Muhammad

Designation: Assistant Professor

Email: abubakr@lums.edu.pk

Date: 13 May 2014

Table of Contents

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

The authors wish to express sincere appreciation to Mr. Syed Abbas and Dr. Abubakr Muhammad for their mentoring during the course of this senior project and in the preparation of this manuscript. Thanks also to Mr. Bilal Talat for his valuable input on handing over the project, and giving us a kick start in understanding its working for us to build upon.

Chapter 1

PROBLEM STATEMENT

Landmine detection systems are crucial in demining hazardous abandoned minefields. The most crucial task is to ensure an optimal distance of the detector from the terrain: this is a hard problem because essentially a demining system has two major constraints; firstly to keep the end effector closest to the ground for high accuracy of detection, and secondly, to keep it far enough to prevent collision with the ground while scanning. Dealing with the second constraint is much more complex due to the inclusion of sensor noise and actuator uncertainties, which introduce a significant amount of variability in the motion of the arm. The bounds on the errors present an ill-posed research problem that must be tackled with. We aim to use a low cost stereo camera pair for 3D profiling of the terrain, and then servo the robotic sensor arm with a calculated precision when hovering over the terrain.

Chapter 2

BACKGROUND AND RELATED WORK

2.1: BACKGROUND

There are two types of demining: mine clearance, which is a wartime procedure performed by armed forces to clear a route for troops to pass, using bombproof bulldozers, carpet bombing, and various other techniques, and humanitarian demining, performed in peacetime to remove the dormant yet easily roused threat to life. Humanitarian demining ("demining") is much more exacting, thorough and requires the use of less invasive techniques. Where mine clearance works while taking calculated risks and accepting that once in a while, a landmine may not have been discovered/cleared, the task is much more difficult in humanitarian demining, where the aim is complete clearance, because locals need to be given the guarantee that the area is free of mines; said area being what locals will walk over, dig for agricultural purposes, and perhaps let their children play on. As such, demining is a much more uphill task, requiring the very best in reliable demining techniques. Traditionally, demining has been performed by hand with metal detectors, which is the most effective, most reliable method. However, it is risky, laborious and time-consuming. Of late, rats and dogs have also been used for the purpose, and now mechanical demining equipment is also deployed.

2.1.1: Mechanical Demining Equipment

Mechanical demining equipment is extensively used for mine clearance applications in many militaries around the world. The most popular form of machine used is a heavily armored bulldozer, which simply bulldozes the ground in its path, detonating or overturning any mine in its path, paving the way for military movement. However, this is extremely infeasible for humanitarian demining purposes, because of the very high acquisition and operational costs which are difficult to justify in poor countries where minefields are large and plentiful and funds are not.

2.1.2: Cost-Effectiveness

Due to the nature of humanitarian demining, effective demining solutions are those that can thoroughly cover a given minefield. The most thorough, effective demining solution is currently manual labor. However, deminers are at risk during the demining operation, and the process of sweeping every inch of the area to be demined is extremely time-consuming, which increases the labor costs. The ultimate, projected goal of our project is to reach a solution that is cost-effective to the extent that multiple robots can be deployed, each augmenting each other and introducing redundancy while still matching or besting the costs of manual demining. Because demining is not a time-critical application, the network of robots can ultimately be set to work, and as long as the work is complete at the end, the time taken is not of immediate importance.

2.1: RELATED WORK

Marwa, a demining robot by the Laboratory for Cyber Physical Networks (CYPHYNETS), LUMS, is closest in vision to our project, and perhaps the only robot similar to ours in hardware architecture [7]. Marwa has demonstrated that it is in fact possible to develop a low-cost autonomous demining robot, and with several such robots it is possible to best the costs of employing a force of demining personnel.

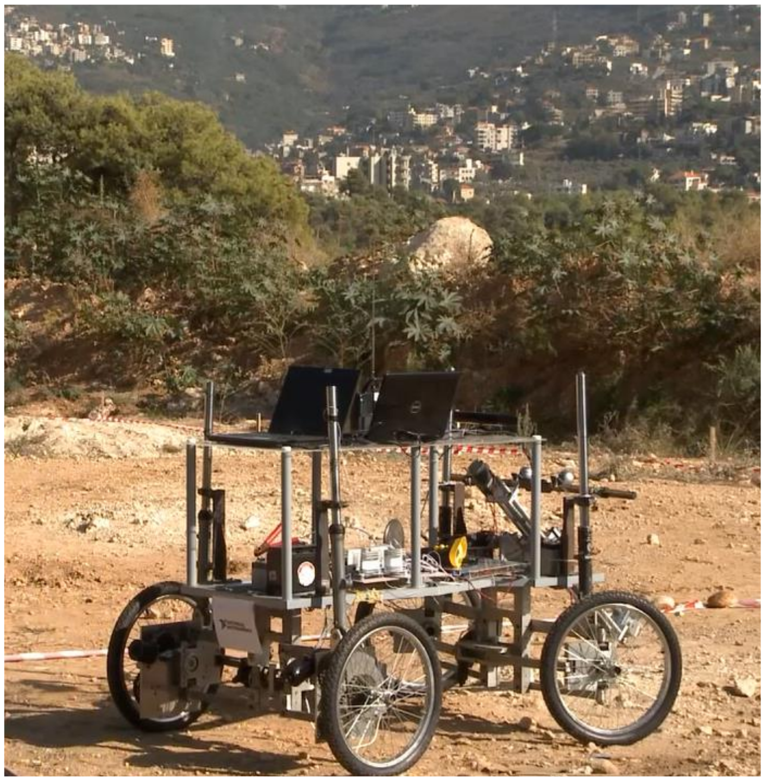
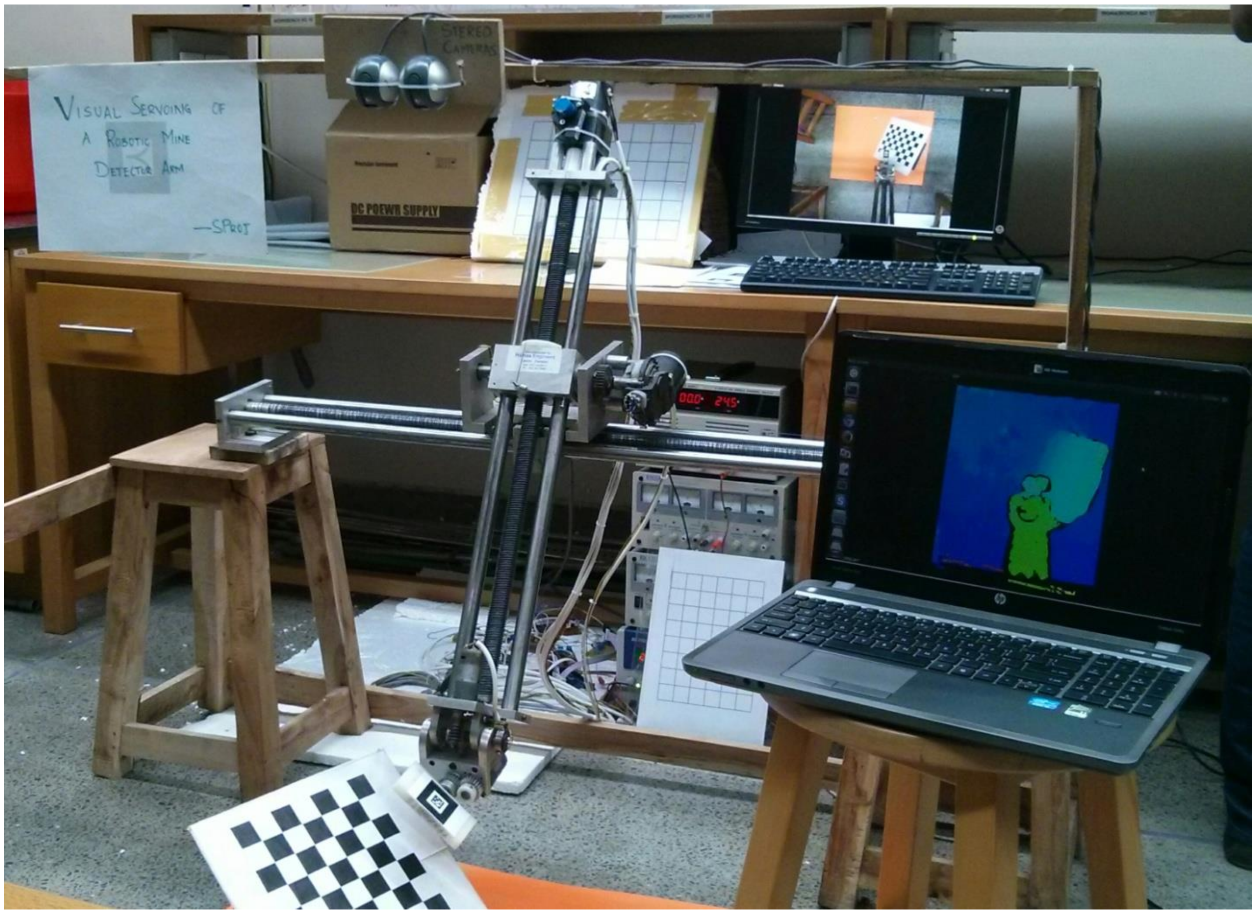


Figure 1. *Marwa* in Lebanon (2011) [7]

Bilal Talat of the same lab has also demonstrated that it was possible to track terrain of realistic configurations with the hardware setup that we have taken as a starting point for our project. However, one critical omission in Talat’s work is that only the position of the end effector has been demonstrated to accurately track a terrain profile in 3D. We have taken care of this omission and demonstrated that not only can such a setup track a terrain profile, but that we can also maintain the end-effector’s pose such that it keeps a mine detector sensor facing the ground as parallel to the ground as possible.

C h a p t e r   3

DESIGN METHODOLOGY AND TOOLS



Front view of the setup

3.1: SYSTEM LEVEL DESIGN

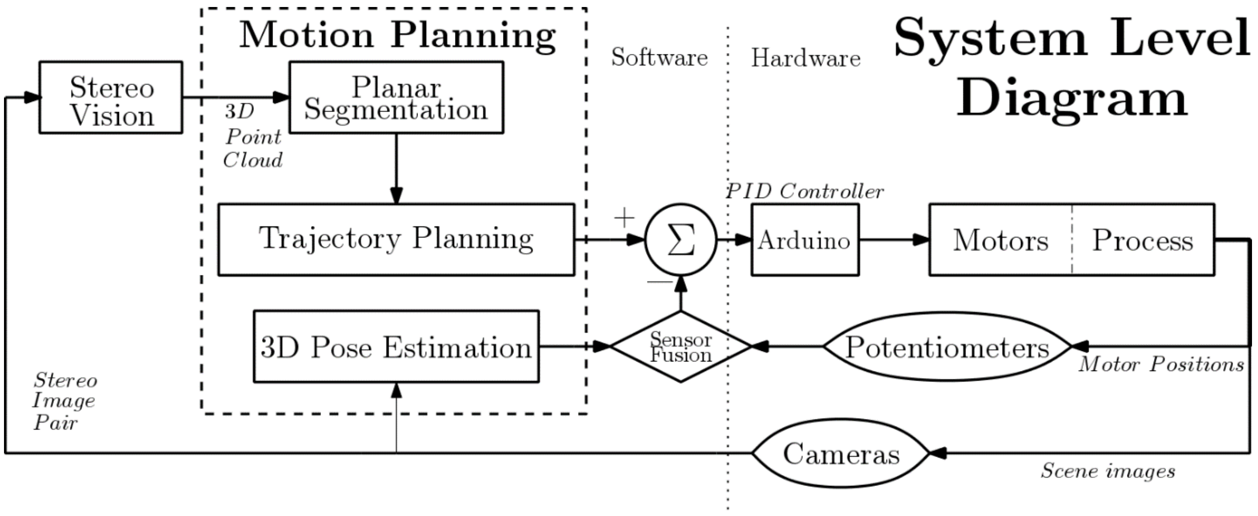


Figure 2. Block Diagram

The overall high level structure is illustrated in the Block Diagram (Figure 2). Our project can be divided into two parts main parts, the Hardware and the Software. The Hardware part consists of the Mechanical Setup including the motors and actuators in the Plant, potentiometers and cameras as the sensors, Arduino as the PID controller, H-Bridge PCBs, and a physical frame to house the arms and the cameras. The Software part consists of Stereo Vision block, a Motion Planner which has Planar Segmentation, Trajectory Planning and 3D Pose Estimation blocks. The actual setup looks like as shown in Figure 3.

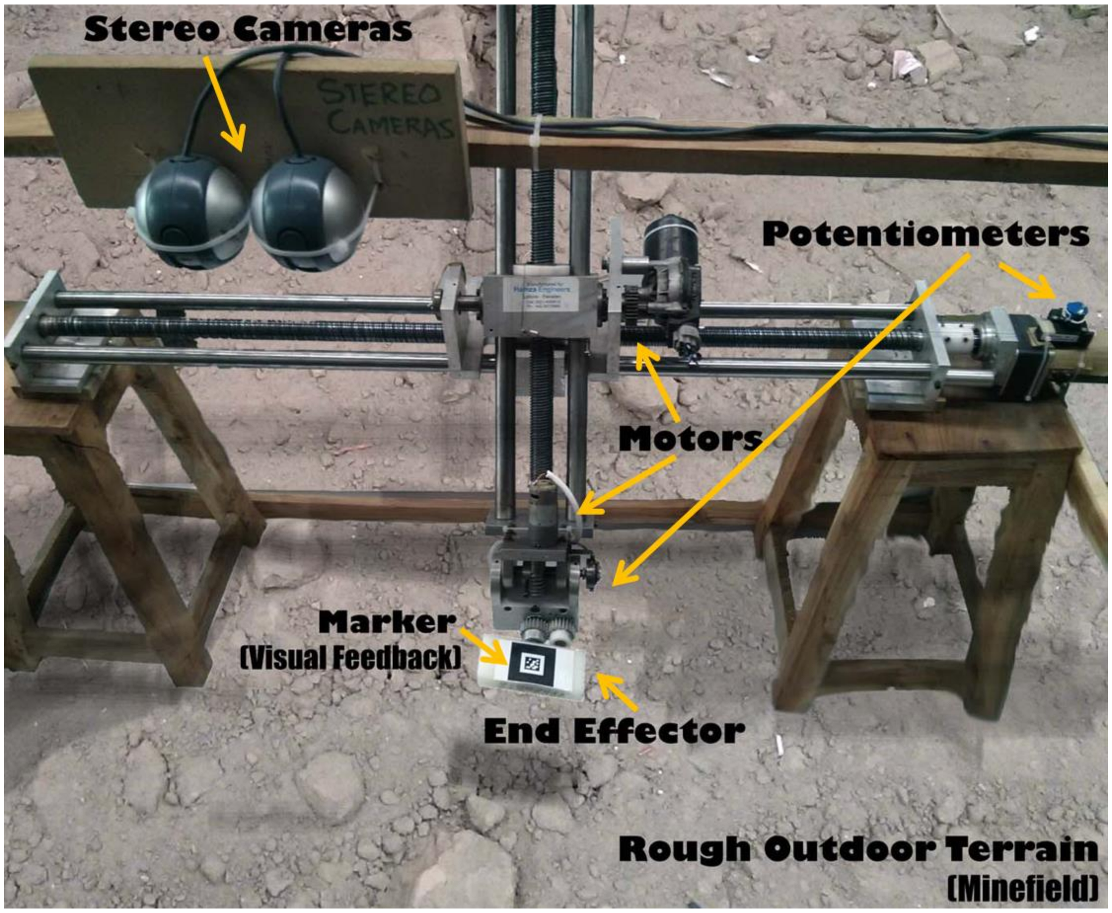


Figure 3. Physical Setup

3.1.1: Mechanical Assembly

3.1.1.1: Degrees of Freedom

Our hardware setup consists of a five degree of freedom robotic mine detector arm. Intitially, at the start of the project, we were handed over a 4 DoF arm. We added a 5th degree (wrist roll) to it. The arm can move horizontally about an axis, it can move vertically and can make angular movements about its vertical axis. Additionally, the end effector of the arm has two connected motors making up the wrist pitch and wrist roll movements. These essentially make up the five degrees of freedom for our robotic arm.

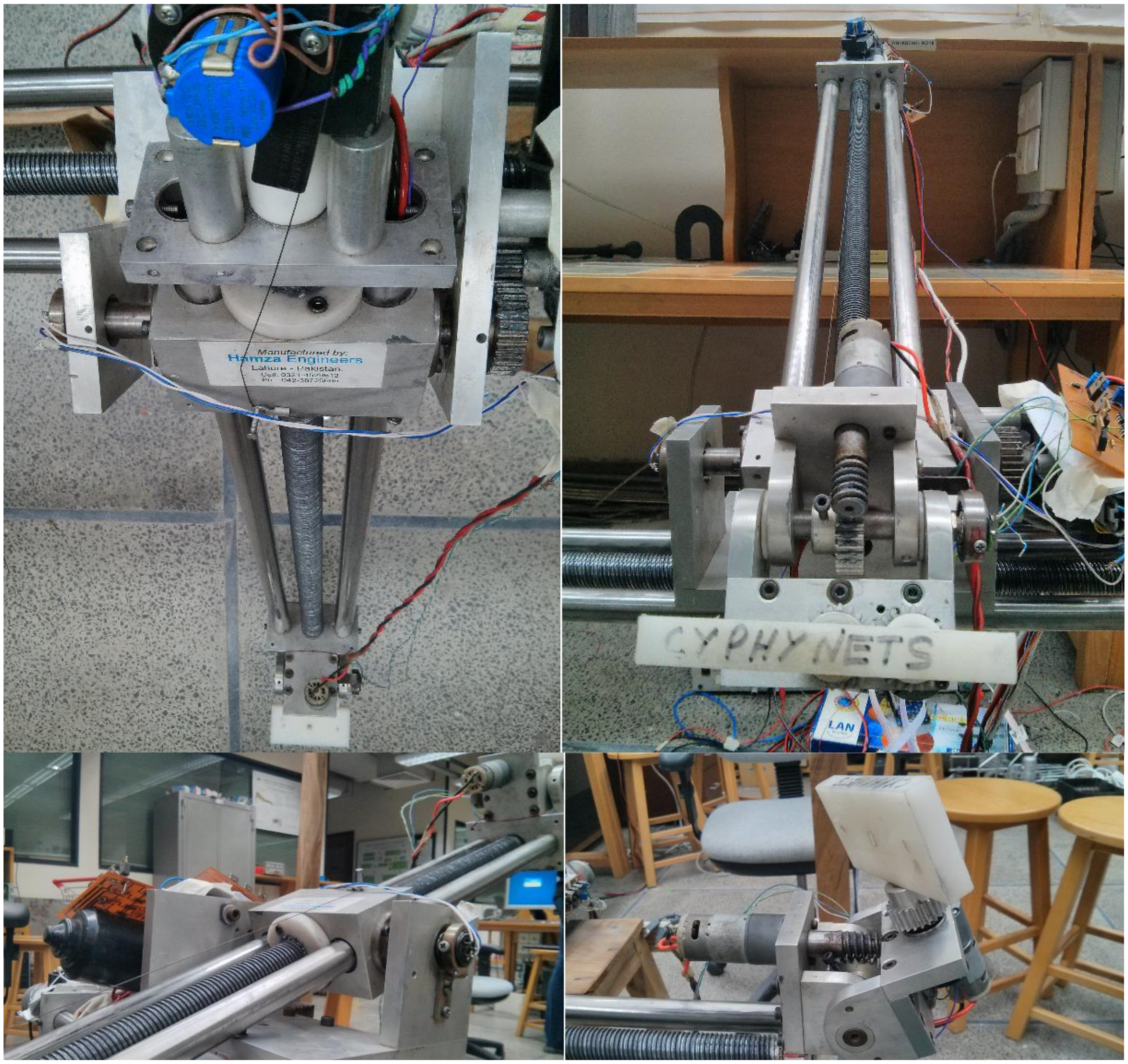


Figure 4. Showing physical limits of the motors

The objective of this project was to use an existing mechanical setup available in the lab as far as possible. This mechanical setup consisted of a 4-DOF mechanical arm designed to be able to reach all points in a cylindrical area in the front of the robot. It was fabricated keeping in mind that any region of terrain can be scanned in multiple different ways, including raster scanning and circular scanning, and the goal was to be able to scan the region of interest with few further actuations and reconfigurations of the arm. Further, taking the assumption that minefields are generally over flat or undulating terrain, noticing that it is thus reasonable to expect that by far the large majority of terrain to be scanned will not have very drastic and very abrupt changes, the mechanical setup was designed such that one prismatic joint lay parallel to the raster sweep path, which allows for the remainder of the degrees of freedom to cater to the irregularities in the terrain while the prismatic joint takes care of the raster motion.

The assumption that minefields are in the vast majority of cases flat and un-rugged is a very reasonable one, simply because the purpose of a minefield during wartime is to ensure that enemy forces are not able to freely charge through an area of land and to provide resistance to hinder their advance. Where trees and thick vegetation is present, as in the case of forests, or jagged and rocky terrain as in the case of mountainous areas, the purpose of a minefield is already being met by the nature in most cases. It is normally only on flat land that the need arises to hinder any possible enemy advance which would otherwise go unobstructed, landmines are ideal candidates for the purposes.

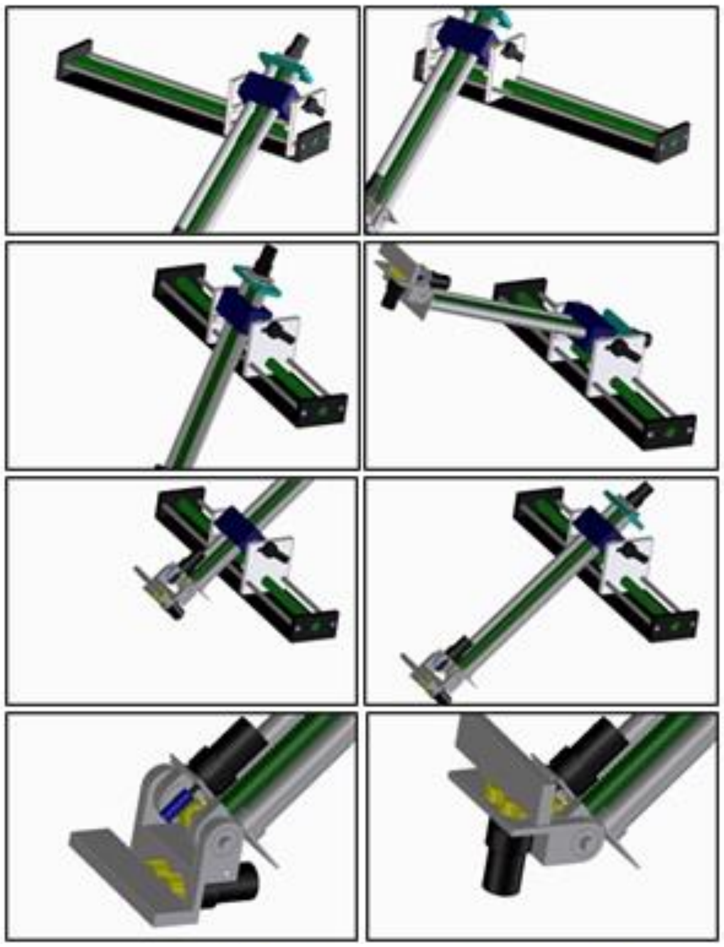


Figure 5. 4 DOF mechanical setup initially available in the lab [7].

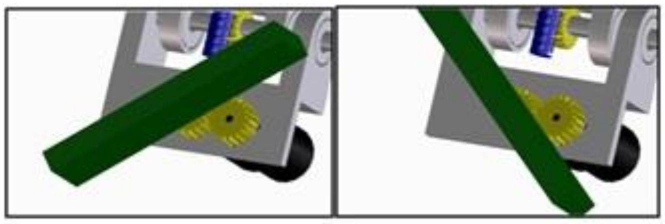


Figure 6. 5th DOF revolute joint [7].

As Figure 5 shows, three further degrees of freedom were incorporated so that the arm would be able to reach all points in a region immediately in front of the arm. However, we found that while this setup was capable of reaching all points in a 3D scan-able region in front of the arm, it was not capable of reaching any arbitrary position within said region with an arbitrary pose, which is essential for the proper completion of this project.

A further revolute joint was introduced at the end effector to take care of this shortcoming. The end result is that the new joint, when actuated in tandem with the previously available joint already installed on the arm, enable the end effector to align itself to a given normal. Figure 6 shows the added joint and the freedom gained from its addition. This brings the overall configuration of the robotic arm to a PRPRR robotic manipulator (P: Prismatic, R: Revolute), with a mine detection sensor attached as the end effector. This mechanism is able to keep the end effector carrying the metal detecting sensors aligned with the ground profile at all times as well as protecting the manipulator from steep or suddenly encountered obstacles in highly uneven terrains.

3.1.2: Software Design

The Software part consists of offline and online subparts. The offline part is only run once before servoing a particular scan region. It is used to generate the 3D point cloud of the terrain for motion planning. 3D point cloud generation uses stereo camera pairs to capture a Stereo Image Pair, that is fed into a Stereo Vision block leveraging OpenCV library to construct a 3D Point Cloud with the use of disparity to calculate depth. With the following underlying equation:

$$Z = fB/d$$

where

Z = distance along the camera Z axis

f = focal length (in pixels)

B = baseline (in meters)

d = disparity (in pixels)

The online part is run at each successive iteration of our scan loop, the dotted portion in the Block Diagram. This block does the entire Motion Planning for the end-effector and consists of Planar Segmentation, Trajectory Planning and a 3D Pose Estimation blocks.

The Planar Segmentation block gets the 3D point cloud of the terrain and segments it into blocks for the Trajectory Planner, which in turn plans a trajectory for the end-effector to follow. It outputs the 3D pose (i.e. spatial, roll, pitch and yaw information) of the end-effector. A 3D Pose Estimation block is used to provide feedback, using the camera to detect the end-effector with the use of a marker, inferring the 3D Pose. Trajectory and 3D pose information of the workspace is transformed into the configuration space using inverse kinematics. The error between the trajectory to follow and the actual trajectory followed is estimated and fed to the controller to close the loop.

A thing to note here is that we align the workspace (Trajectory and 3D pose) with a universal axis that we define to be the robot axis, to maintain consistency of our data. Refer to the Section 3.2.1.2.

3.1.3: Hardware / Software Interface

We interface our hardware (motors and potentiometers etc) with the motion planning module (software codes) through an Arduino Mega 2560 microcontroller as mentioned before. It has 15 PWM outputs which are sufficient for the 5 DoF movements through H Bridges. Additionally, it has 16 analog inputs, 4 UARTS and a 16 MHz crystal oscillator. The Arduino controller receives values from the motion planning part and forwards it to the motors through 5 H Bridges for 5 different motors to reach a desired place. It also returns the values from the sensors (potentiometers) to the motion planning module via serial communication.

Figure 7. Arduino Microcontroller

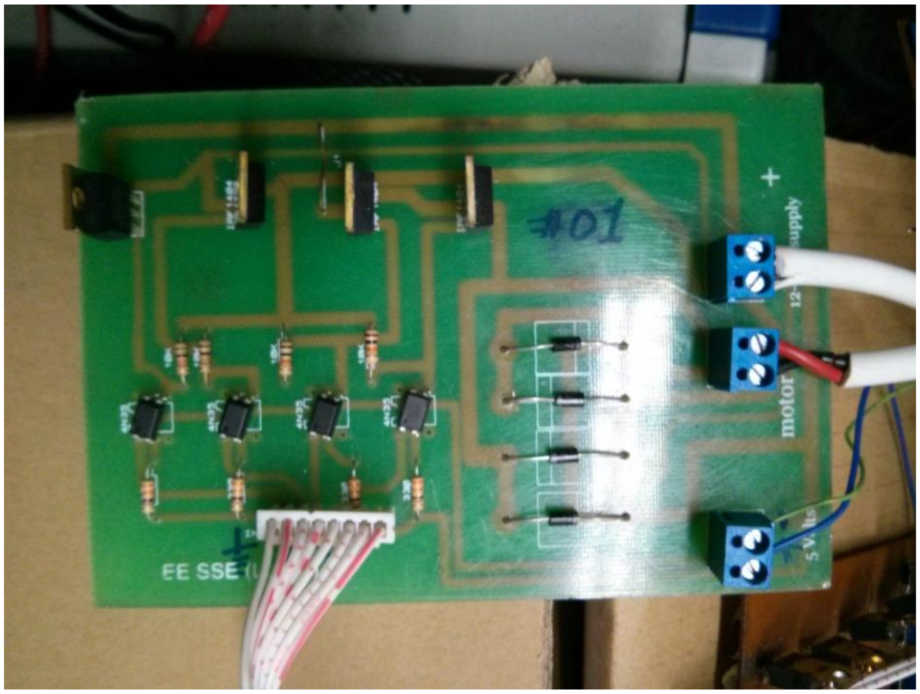


Figure 8. H Bridge Design

3.1.4: Serial Communication

The serial communication between the Arduino and the code is done making the use of a library libSerial and its extensions. The values are sent and received instantaneously without much delay and the system moves according to the current values being sent and received. This results in efficient execution of the motion physically through motors, which is initially planned in the code only.

3.2: DESIGN METHODOLOGY

3.2.1: Calibration and Alignment

3.2.1.1: Stereo cameras

Firstly, we need to calibrate our vision system. This is a two-step process. In this first step, we calibrate the individual cameras. We do this using a chessboard of size 8 x 6. OpenCV library [1] has built-in camera calibration sample codes; we use one of those. Chessboard being planar surface, with known square size (e.g. 3cm by 3cm) is an easily available candidate for this step.
So, individually for both the cameras, we capture a series of chessboard images placed at different positions and orientations in the image. The algorithm detects chessboard corners up to sub-pixel accuracy and saves their pixel locations for each image (See Figure 9). Then, it finds a 3 x 3 projective transform to map the corresponding points of the (8 x 6 = 48) chessboard corners minimizing a cost function (e.g. least squares estimation). In the end we have the camera matrix having the focal length, unit-of-length to pixel ratio and principal point offsets.

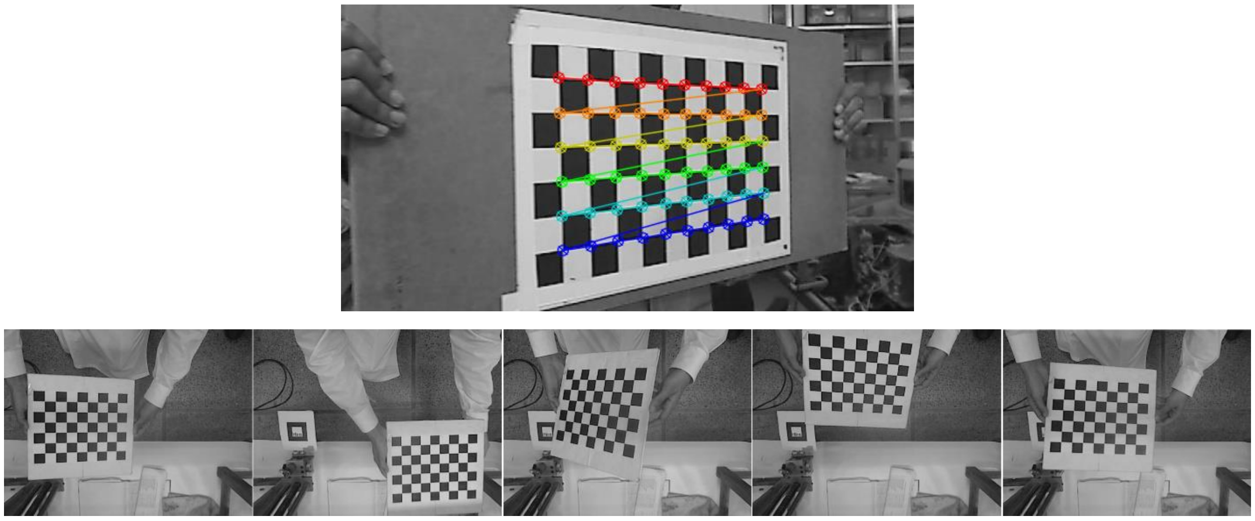


Figure 9. Camera calibration images set

Using the camera matrices for both cameras and the same set of calibration images, we find one-to-one mapping between the corresponding images from both cameras, and then estimate a 3D transformation existing between both camera centers and the frames of reference. This matrix then corresponds to the Essential Matrix for the stereo pair: it saves the relative geometry existing between the two cameras. Image pair can now be rectified (Figure 10).

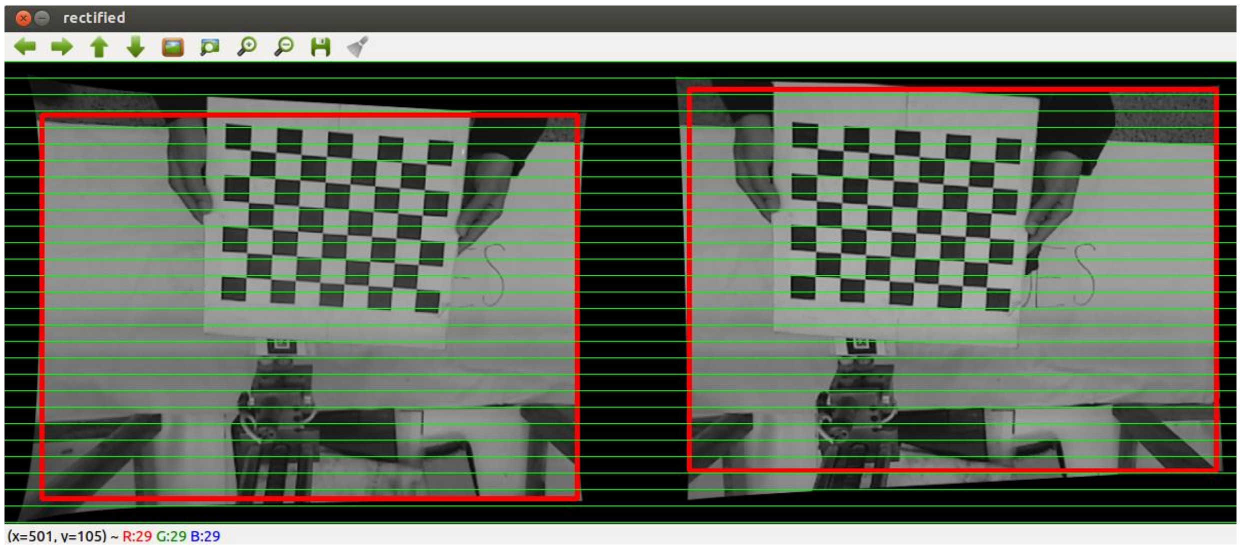


Figure 10. Rectified image pair after stereo calibration

The cameras we are using are simple Logitech webcams (should be identical, see Figure 11); this calibration step though, has been performed on other sets of stereo cameras (Logitech C900 HD webcams) too.

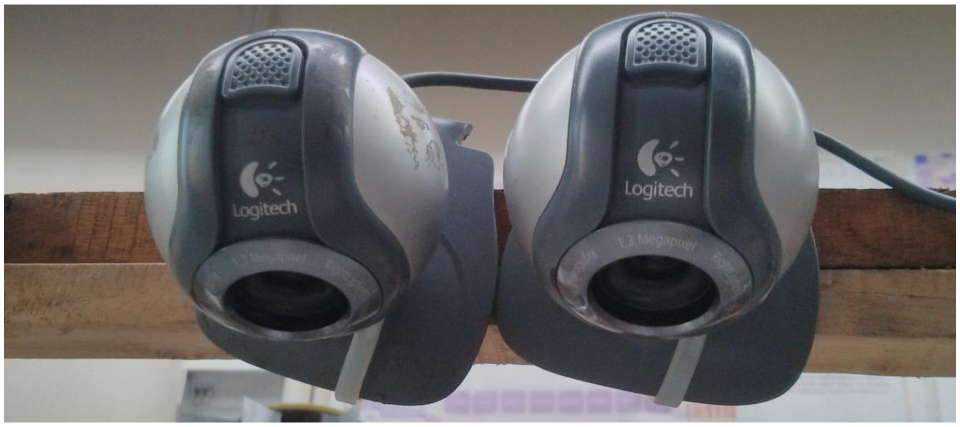


Figure 11. Stereo camera pair

3.2.1.2: Motor / Camera reference frames

Because of natural mismatch in the reference frames of the camera and the motors, we need a rigid body transform to apply on our generated terrain profile so that:

i. Camera frame's x-axis aligns with the horizontal prismatic joint's axis
ii. Camera xy-plane should be parallel to the world flat ground (i.e. z-axis of camera and motor frames should be parallel)

For this we needed to apply a 3D rotation on all data processed through camera images, in getting depth information (Figure 12 and Figure 13 illustrate this rotation).

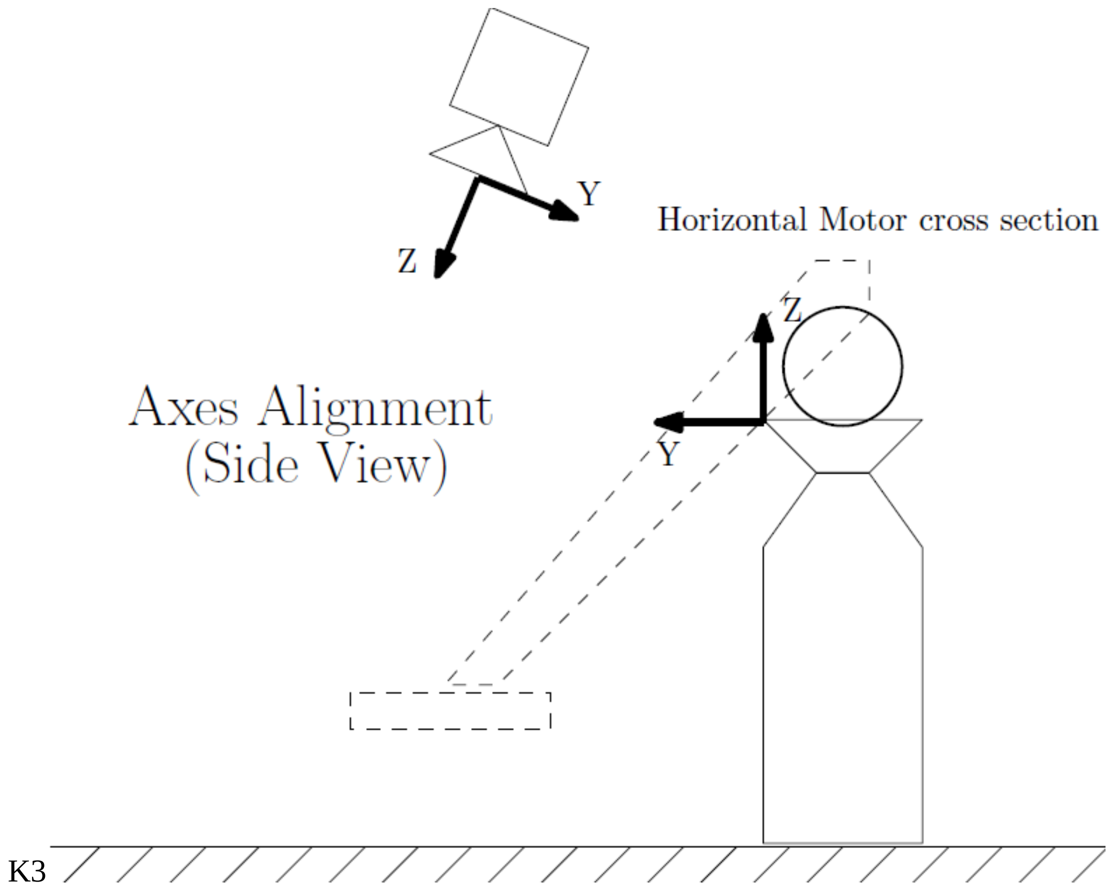


Figure 12. Aligning camera xy-plane (i.e. z-axis) with flat ground

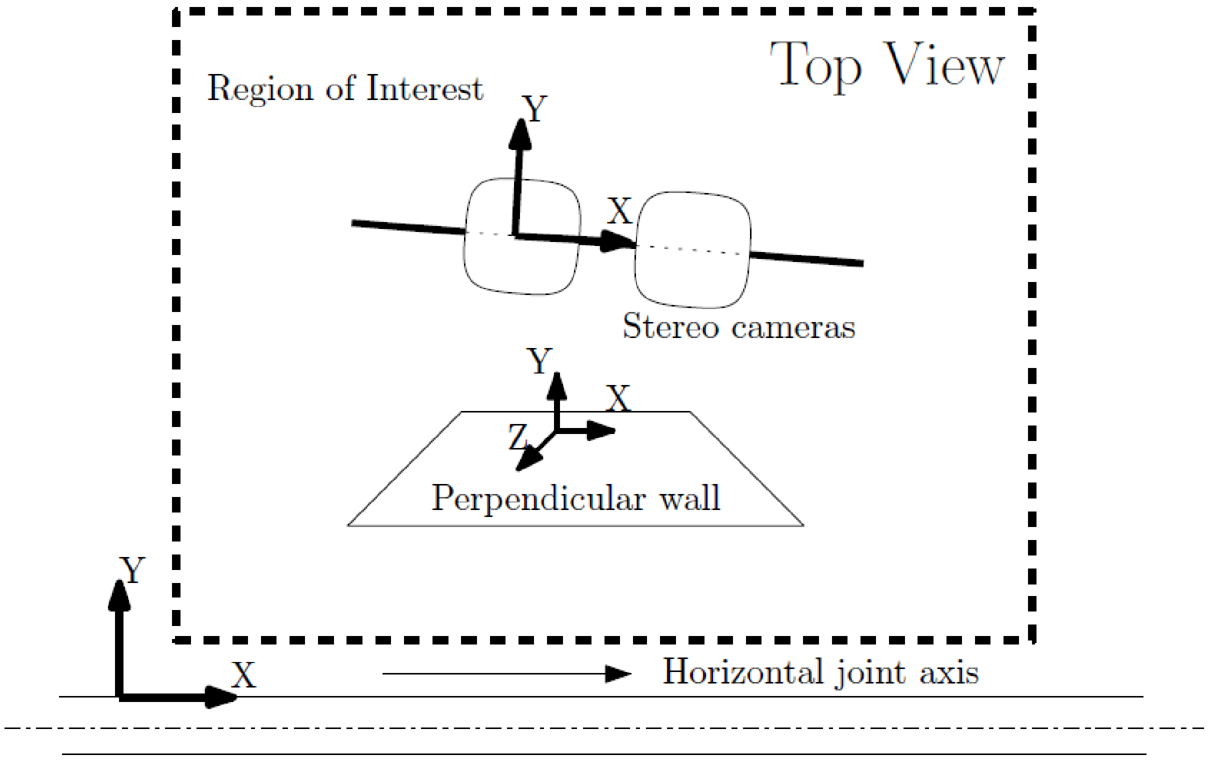


Figure 13. Aligning x-axis and motor axis

To achieve this, our algorithm is as follows:

i. Once we have the point cloud generation step running, we capture an image having flat ground as well as a wall (not necessarily perpendicular to the ground) whose normal vector is perpendicular to the horizontal joint axis.

ii. We manually select these two regions in image, and generate separate point clouds.

iii. Now we fit a plane for each of the two flat surfaces (the wall and the ground).

iv. Using the normal vectors of the planes, we create a unique 3D rotation matrix such that:

   a. Ground normal vector equals camera z-axis

   b. Wall normal vector equals camera x-axis

v. Optionally, we also add the translation vector in our transformation matrix, and save this as a *“transformation.yml”* file. This will be later used for terrain profiling step.

3.2.2: Vision System

3.2.2.1: Generating point cloud

We use OpenCV to generate the terrain profile from our stereo vision system (Figure 14). The algorithm employed works as follows:

i. Capture the image of the terrain (e.g. minefield) from cameras.

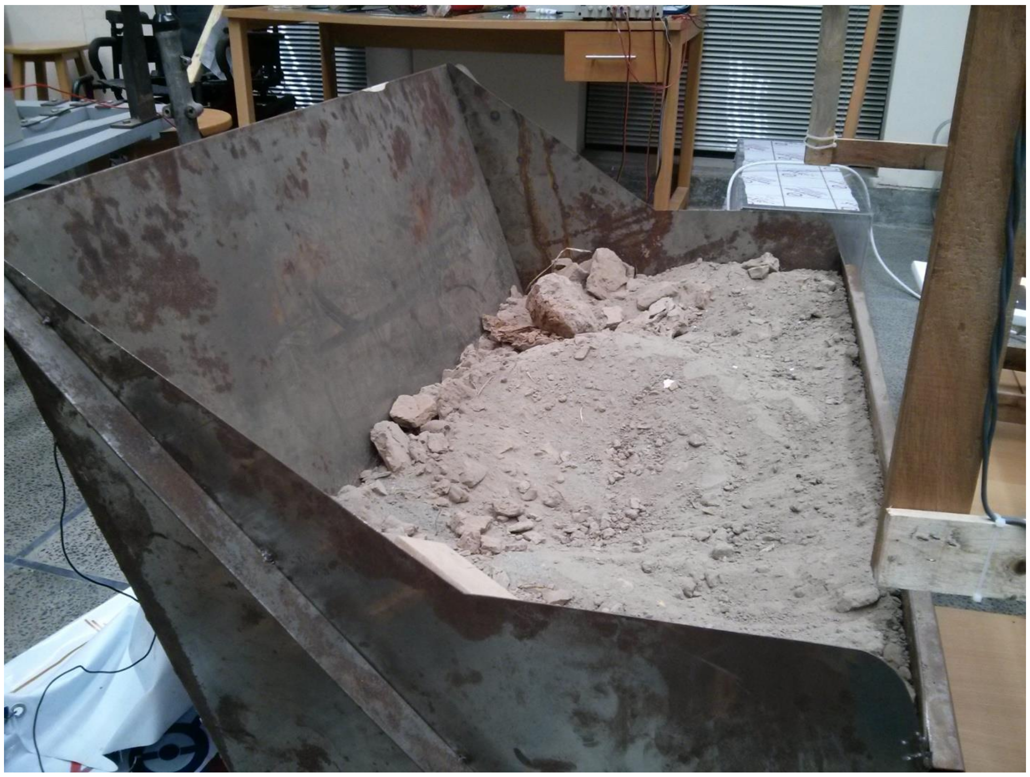


Figure 14. Rough Terrain

ii. Use Semi-global Block Matching technique to find correspondence pairs in both the images (built-in OpenCV functions).

- SGBM [2] technique is a robust matching technique for stereo correspondences as it works quite well for outdoor environments (radiometric differences), and is also faster as compared to highly-accurate graph-cuts method.

iii. This generates a *disparity map* of the captured scene. This map is the product of all stereo processing: it contains pixel-wise depth information.

iv. Next, specify a region of interest (ROI) in your image.

- ROI is defined, crudely, for all the area covering the image in which our degrees of freedom of actuators allow the end-effector to reach.

v. Lastly, we use this map as well as the extrinsic camera parameters containing relative geometry of cameras (from calibration step), to create a 3D map, i.e. for each pixel in the disparity map we reproject it in 3D to get (x, y, z) co-ordinates of surfaces.

- This 3D map is saved as a point cloud data file standardized in the Point Cloud Library (PCL) [3] file (Figure 15 and Figure 16).

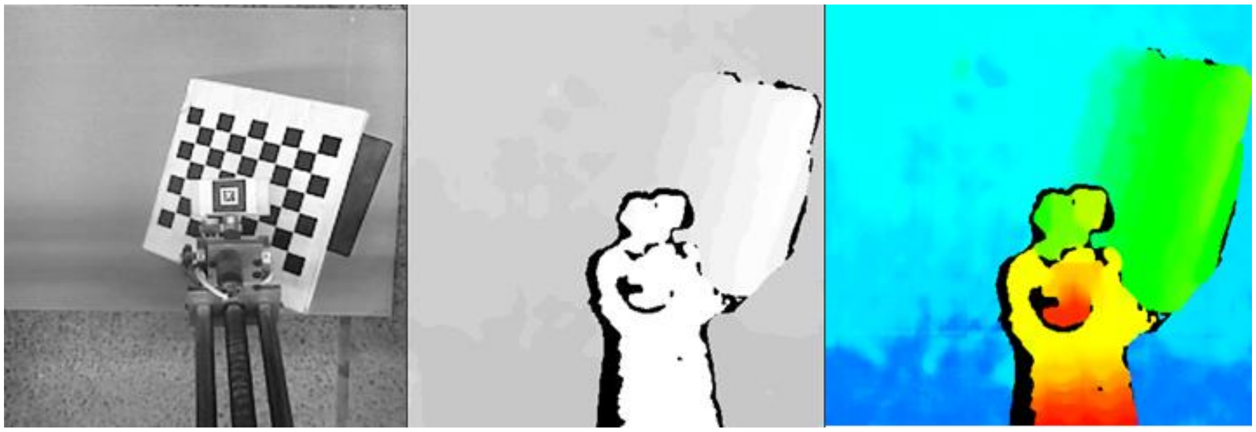


Figure 15. Scene image, its corresponding disparity map, and its 3D point cloud

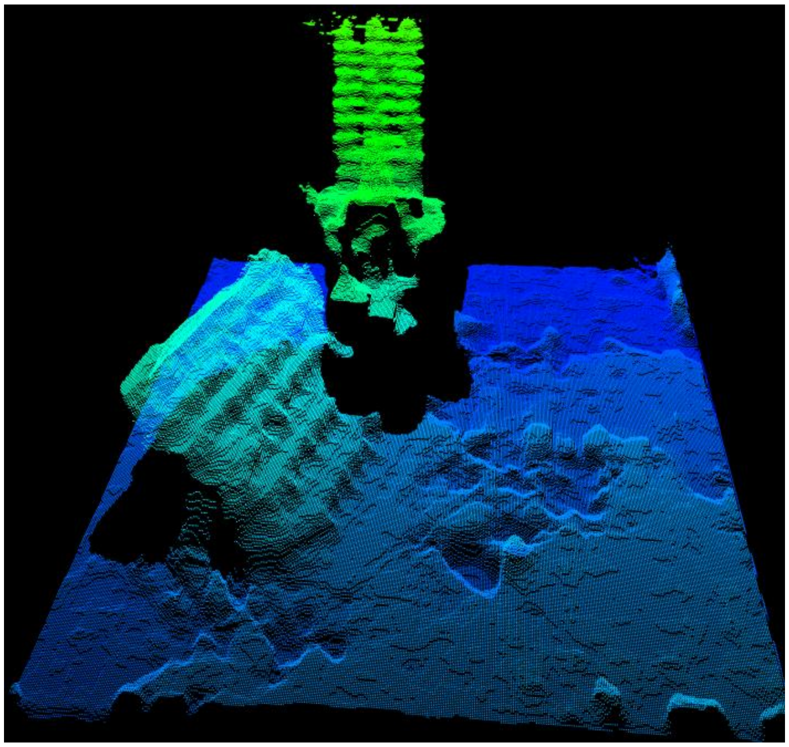


Figure 16. Point cloud of scene from Figure 15

3.2.2.2: Axes Alignment

Once we have the point cloud, we use our initially generated transformation matrix (transformation.yml) to transform our points so that both reference frames get aligned. Once aligned, before processing further, we remove the outliers from the cloud using a built-in statistical outlier removal filter.

3.2.3: Planar Segmentation

Once we have the aligned point cloud we have to do planning of our end-effector for maneuvering over the terrain. Here, we apply planar segmentation strategy. In this, we segment the whole cloud into rasters to create a grid. In each grid cell we have set of 3D points. The grid cell size is variable and to achieve better accuracy we use smaller sized grids (~ 5cm by 5cm). Now, in each grid cell we fit a plane. The planes are fit using RANSAC (RANdom SAmple Consensus) [4]. We use PCL library functions for plane fitting. For each plane we save the four parameters per grid cell, and store this whole discretized map for further planning.

3.2.4: Trajectory Planning

Once we have the planes equations, we have to plan the motion of our end-effector such that it remains closely parallel to each plane (in the grid), as well as at a certain safe distance to achieve close tracking (Figure 17).

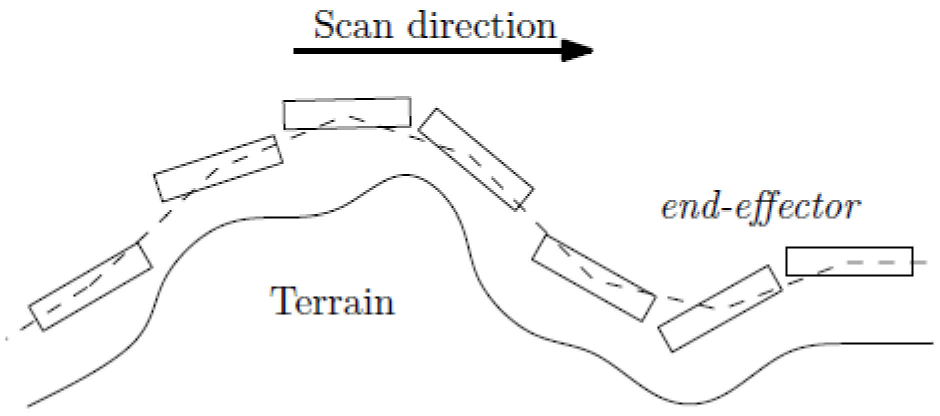


Figure 17. Scanning strategy

Using the plane equations, we determine a discretized terrain map. As a scanning strategy, we employ simple raster scanning over the grid, but for that we need to find the trajectory. For each set-point in the terrain profile, firstly we find the exact co-ordinates in 3D of our end-effector. For this, we use the plane's normal vector and along that normal just at a safe distance we determine a 3D point (x, y, z). Secondly, we have the plane normal vector (x_n, y_n, z_n).
With these six work-space co-ordinates, we find the corresponding five configuration-space motor co-ordinates (note that our system has five DOF), i.e. the three angles for three revolute joints, and two distances of the prismatic joints. This step, is explained in detail in the section on Inverse Kinematics. With this we create a trajectory of motor co-ordinates which would later be used while execution of the operation.

3.2.5: Inverse Kinematics

The inverse kinematics module is crucial in making sense of sensor inputs received from the arm and the pose of the end effector determined by the stereo vision system. Given any arbitrary pose, the inverse kinematics module determines the best-fit configuration of the arm that will result in a pose as close to the input pose as possible, given the minimum actuation amount of the motor actuators. This transformation from workspace to configuration space is then used to determine how next to actuate each of the manipulator joints so that the next target pose in the planned path of the end effector is achieved.

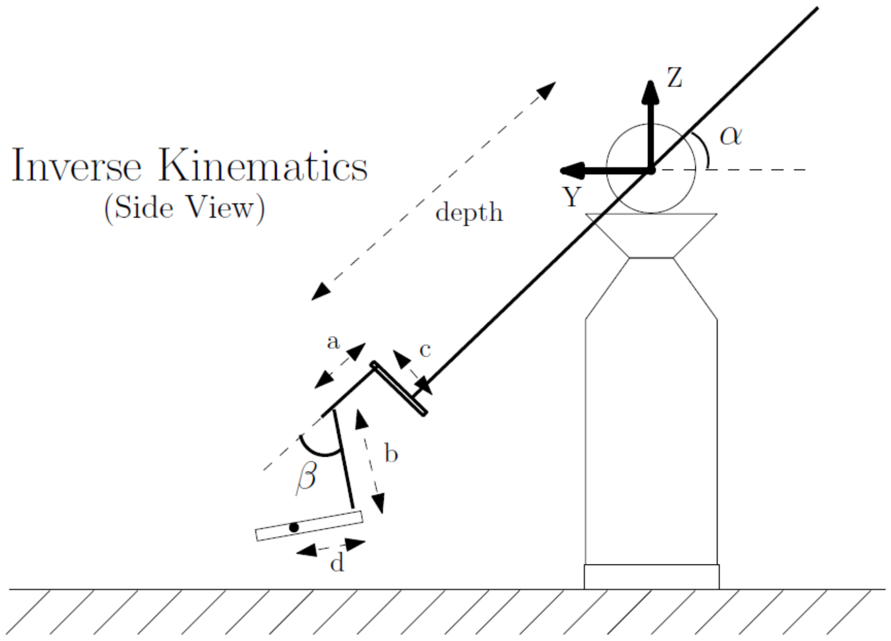


Figure 18. Side profile of manipulator arm

It is instructive to note that if the horizontal sweep prismatic joint is considered to be along one axis, the x-axis in our case, it becomes decoupled from the remainder of the four joints and the parameter along that axis is described by $x = x$. The remainder of the four joints are coupled and determine the position of the end effector in the y-z plane. We label the manipulator's parameters as illustrated in Figure 18. Then, given a workspace position vector, $v = [x, y, z]$ and a normal vector, $\bar{n} = [x_n, y_n, z_n]$ the following set of equations give the corresponding configuration space parameters α, β and $depth$:

$$\theta = \tan^{-1}\frac{y_n}{z_n}$$

$$\beta = \theta + \frac{\pi}{2} - \alpha$$

$$y' = (depth + A + B * \sin\beta + D * \cos\beta) * \cos\alpha + (C + B * \cos\beta + D * \sin\beta) * \sin\alpha$$

$$z' = (depth + A + B * \sin\beta + D * \cos\beta) * \sin\alpha + (C + B * \cos\beta + D * \sin\beta) * \cos\alpha$$

The internal workings of the inverse kinematics module are as follows. Given an input pose (position and end-effector normal vector in 3D Euclidean space), the algorithm searches the entire configuration space heuristically to find the configuration whose pose has the closest distance from the input pose. The search algorithm initially sweeps the entire configuration space in a coarse sweep which is still fine enough to ensure that the next step will guarantee recursion into regions that will contain the closest configuration, and then recursively searches the vicinity of the most promising configurations (those configurations with the least 'distance' from the input pose). It then returns the closest configuration found i.e. the one with the least heuristic distance.

The heuristic distance, d is a weighted combination of the Euclidean distance between the end-effectors in search configuration and the input configuration, and the cross product of the normal vectors of the given input configuration and the search space configurations:

$$d = \lambda_1\sqrt{(x - x')^2 + (y - y')^2 + (z - z')^2} + \lambda_2 \| [x_n, y_n, z_n] \times [, \sin\theta', \cos\theta'] \|,$$

Where θ' is the value of the wrist angle returned by the algorithm in the closest configuration, and λ_1 and λ_2 are tuning parameters

3.2.6: 3D Pose Estimation

ARToolKit is based on a basic corner detection approach with a fast pose estimation algorithm

The computer vision algorithm steps are as follows:

i. Original Image

ii. Threshold Image
iii. Connected Components
iv. Contours
v. Extracted marker edges and corners

We modify the original step of thresholding beyond a fixed value and do dynamic thresholding in its stead. This is done to incorporate the various environments in which the robot may need to operate, it further improves the results by mitigating the light intensity variations inherently associated with our operation of tilting and translating the end-effector.

The estimated pose is in the camera coordinate frame, we transform it to the Universal Robot Axis to maintain consistency with our calculations.

3.3: TOOLS

3.3.1: Arduino Microcontroller

We use an Arduino microcontroller (Arduino Mega 2560) for serial communication of values to be given to the motors and the values that the sensors (potentiometers) return.

3.3.2: PCL/OpenCV

We use PCL and OpenCV libraries for running the stereo pair and generating point clouds at different stages. We do all the coding in C++. We make use of different libraries inside OpenCV and PCL. The RANSAC algorithms are also applied using these libraries for the plane fitting.

3.3.3: ARToolKit

ARToolKit developed by the University of Washington was used under the GNU General Public License [5]. The toolkit was leveraged to estimate the 3D pose of the end-effector.

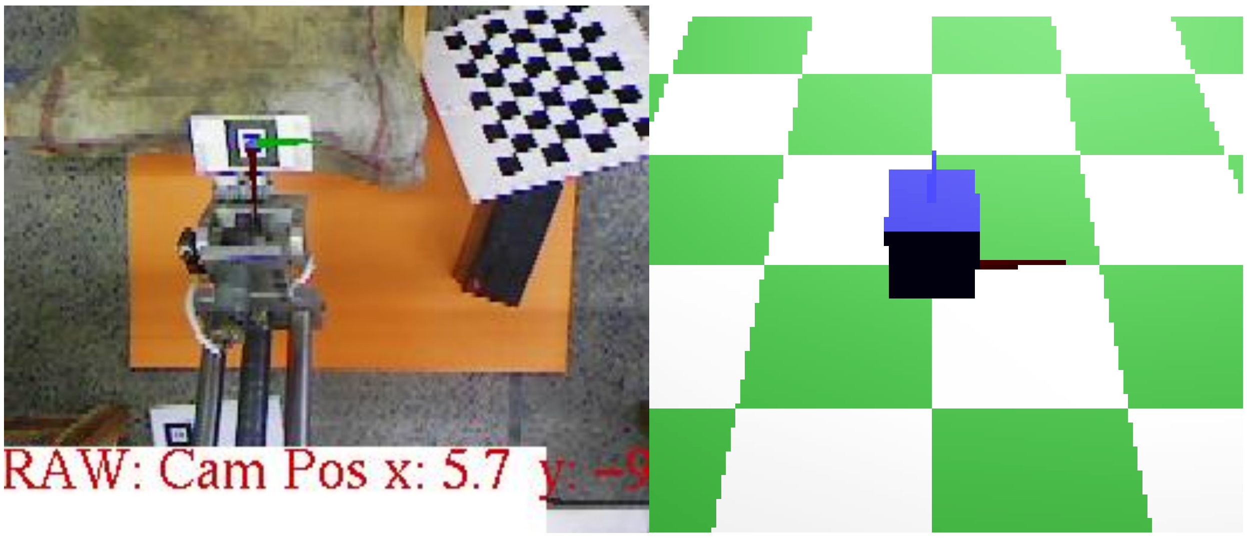


Figure 19. Showing end-effector detection
using vision

Figure 19 on the left shows the physical end-effector with a marker placed on top, on the right is the software representation of the end-effector estimated pose.

A sequence of steps are followed to first calibrate ARToolKit with our camera. We follow the Two Step Calibration approach mentioned in the ARToolKit documentation to calculate the camera distortion and then calibrate our camera. The errors were within one millimeter.

We use the Hiro Pattern that comes with the toolkit and paste it on our end-effector for 3D pose estimation. The toolkit uses pattern matching techniques to locate the end-effector in the scene and estimate its distance from the camera center along with its translation along the other axes. The tilt, i.e. roll, pitch and yaw is also estimated from the tilt of the marker itself.

ARToolKit is based on a basic corner detection approach with a fast pose estimation algorithm Please refer to section 3.2.6 for the algorithm details and section 4.2.3 for some simulation results. The toolkit is fast enough for real time applications, it runs within ~200 milliseconds.

3.3.4: Linux Environment

The Linux environment we used for our senior project is the Ubuntu 12.04.4 LTS (Precise Pangolin, Linux kernel 3.4) OS, and there were multiple factors motivating the decision of OS environment. First and foremost, a Linux environment was needed because all of the other tools we employed throughout the course of our project supported either only Linux, or both Windows and Linux, in which case Linux was given the nod because of greater control over communications ports (USB/Serial), pre-installed and pre-configured GCC complier and environment for C++ programming, and fewer background processes to clog the scheduler when running our program. The long-term-stability (LTS) release was chosen because of its maturity as a platform, extensive user community and support forums, and its extended support period which ensures that timely fixes are released for any bugs encountered, and support is available to deal with any erratic behavior.

Chapter 4

IMPLEMENTATION RESULTS AND LIMITATIONS

In this chapter, we will be presenting the results obtained so far during the course of this senior project from implementation perspective.

4.1: HARDWARE CHARACTERISTICS

4.1.1: Motor Characteristics/Current Ratings

There was play in the angular motor (angular movement of the arm) which hindered efficient traversing of the path planned. The motors drew a lot of current. The 5 motors required 12-15 amps of current to run in synchronization in an efficient manner. Initially the power supply available couldn’t supply this much amount of current (only 3 amps max.) and hence jerky movements were observed. A 30 volt 10 ampere power supply was then purchased and the motors worked fine. The issue of starting current which was often very high was also solved by the use of the new power supply.

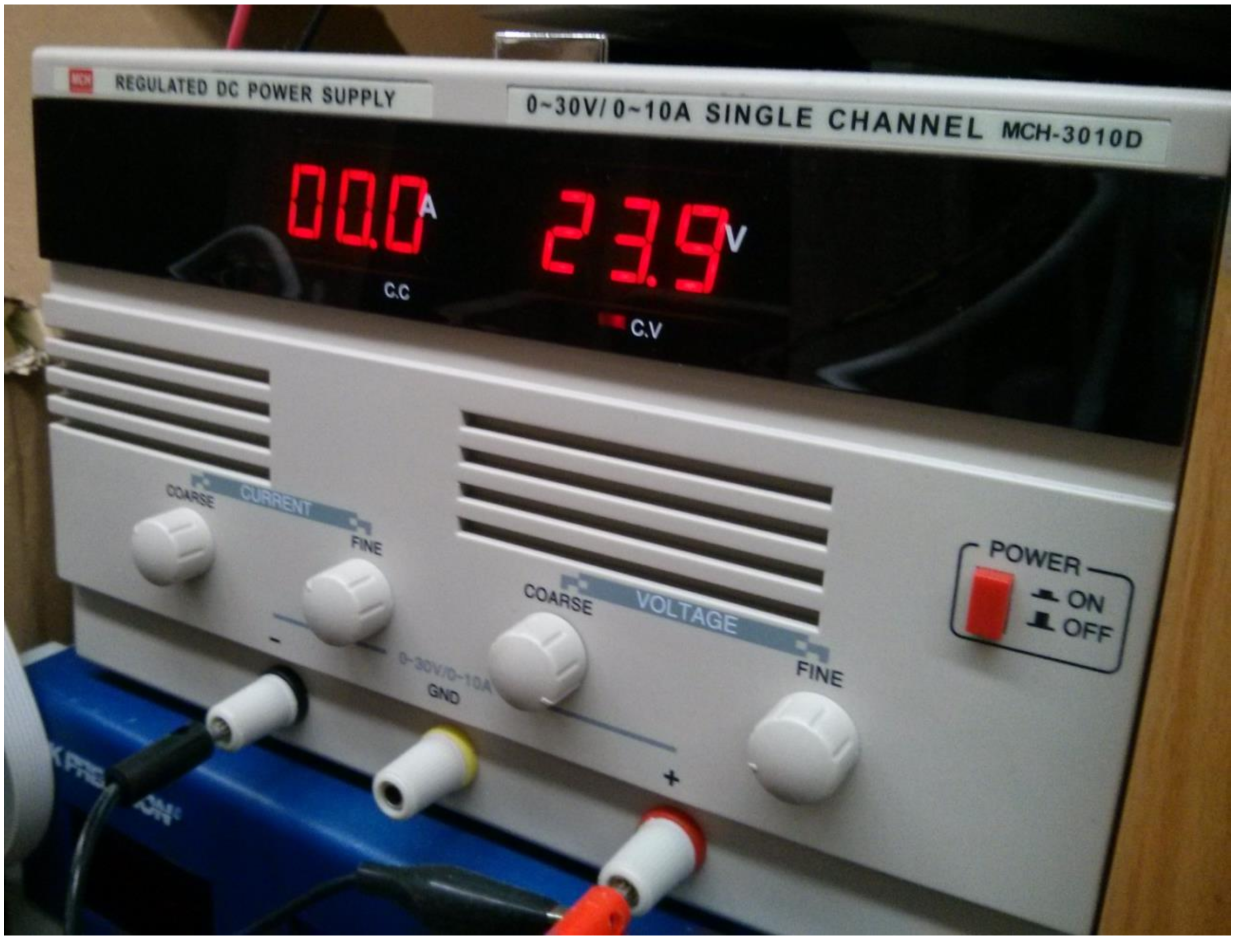


Figure 20. New Power Supply (30 V / 10 A)

4.1.2: Sensitivity of the Motors

Motors are sensitive to the increments in units of angular and linear position shown in Table 1.

Table 1. Motor Characteristics

Motor	**Sensitivity**	**Power Rating**
Horizontal	1.15 cm	24V/3A
Depth	0.6 cm	24V/3A
Arm angle	7.2 degrees	24V/3A
Wrist Pitch	1.5 degrees	5V/0.1A
Roll	2.5 degrees	5V/0.1A

4.2: IMPLEMENTATION RESULTS

4.2.1: Point Cloud Data

For a series of terrain profiles, we managed to generate point clouds and plan trajectories of the end-effector over it. Images from some of those experiments with the images of variable types of terrains and their point clouds over which we planned trajectories and executed operation are shown.

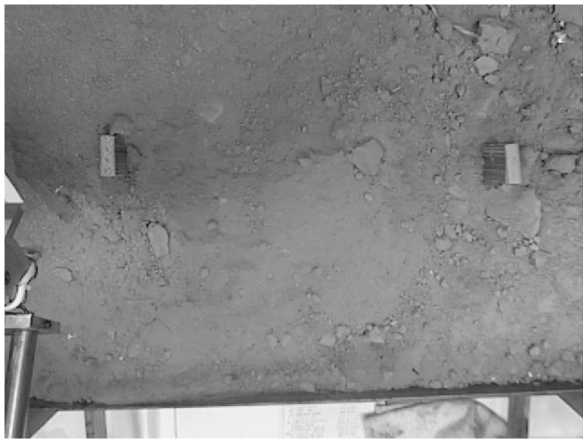


Figure 21. Scene image (test # 1)

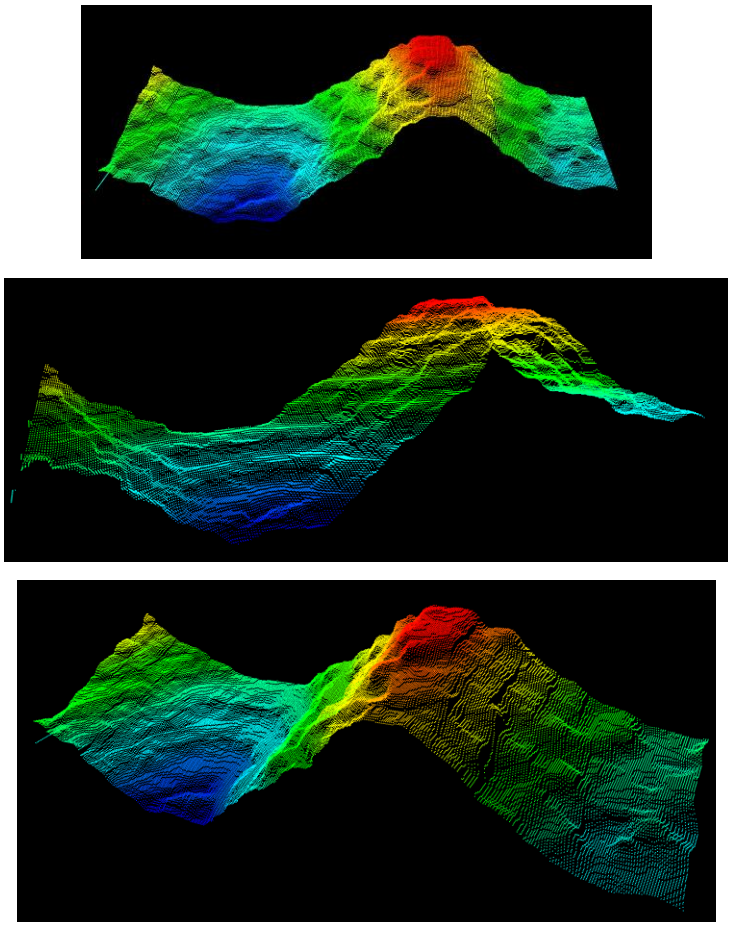


Figure 22. Point Clouds (test # 1)

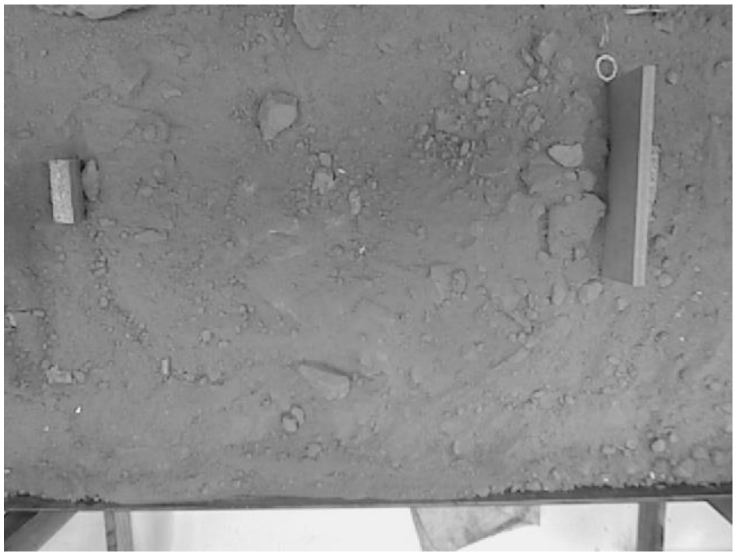
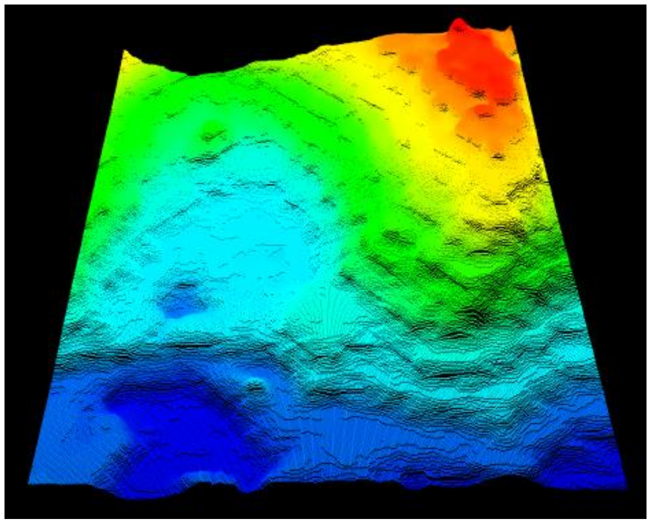
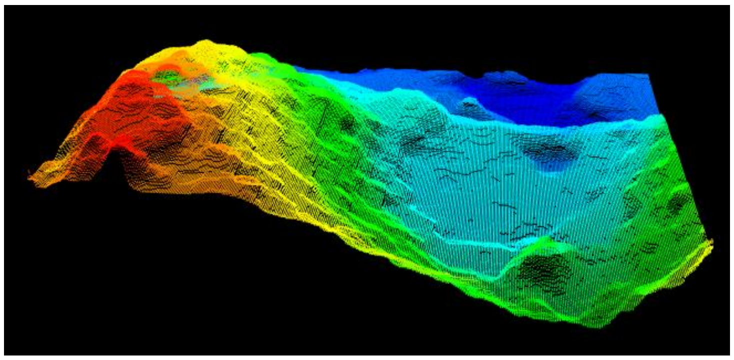


Figure 23. Scene image (test # 2)





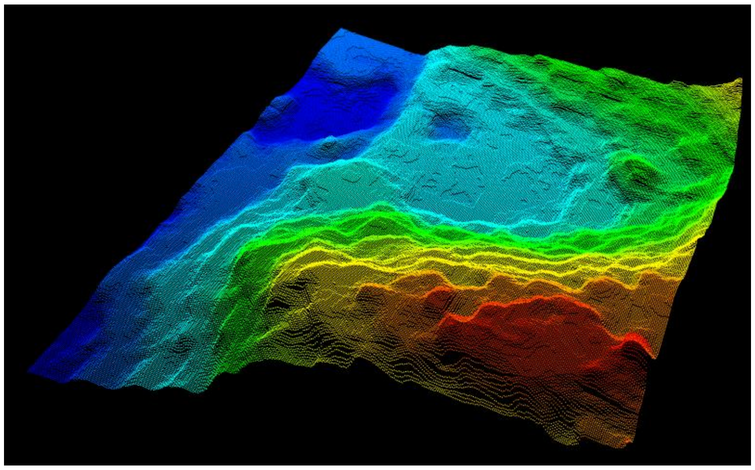
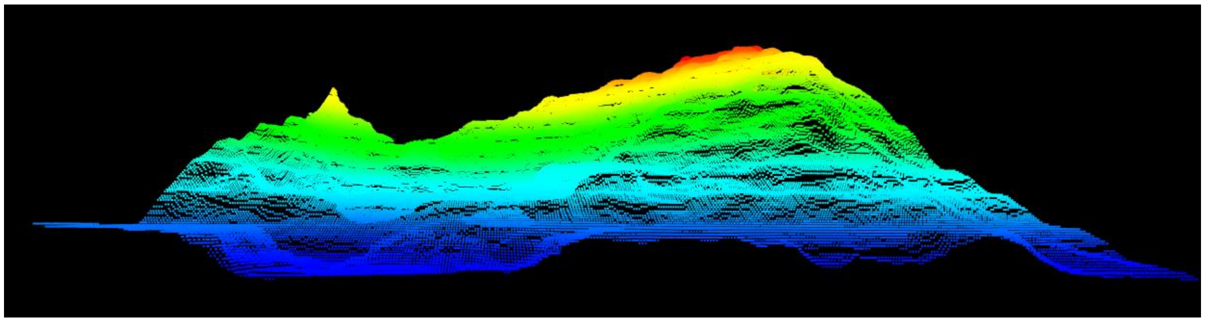



Figure 24. Point Clouds (test # 2)

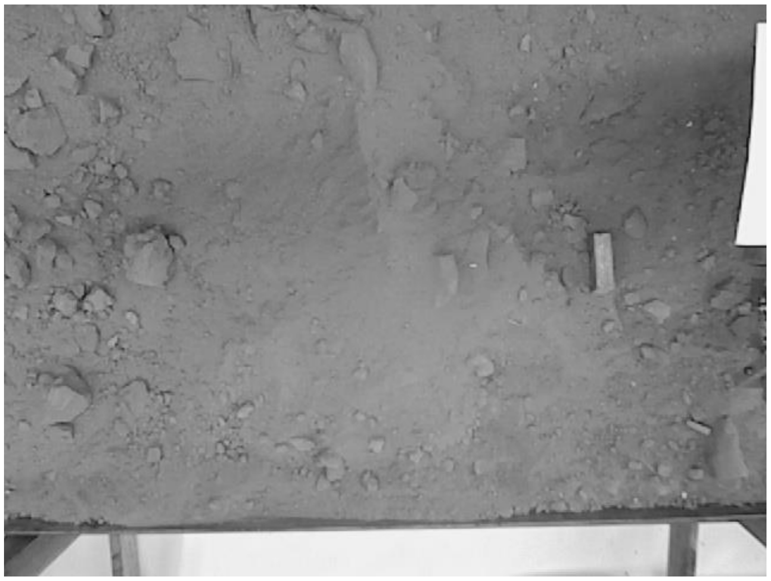


Figure 25. Scene image (test # 3)

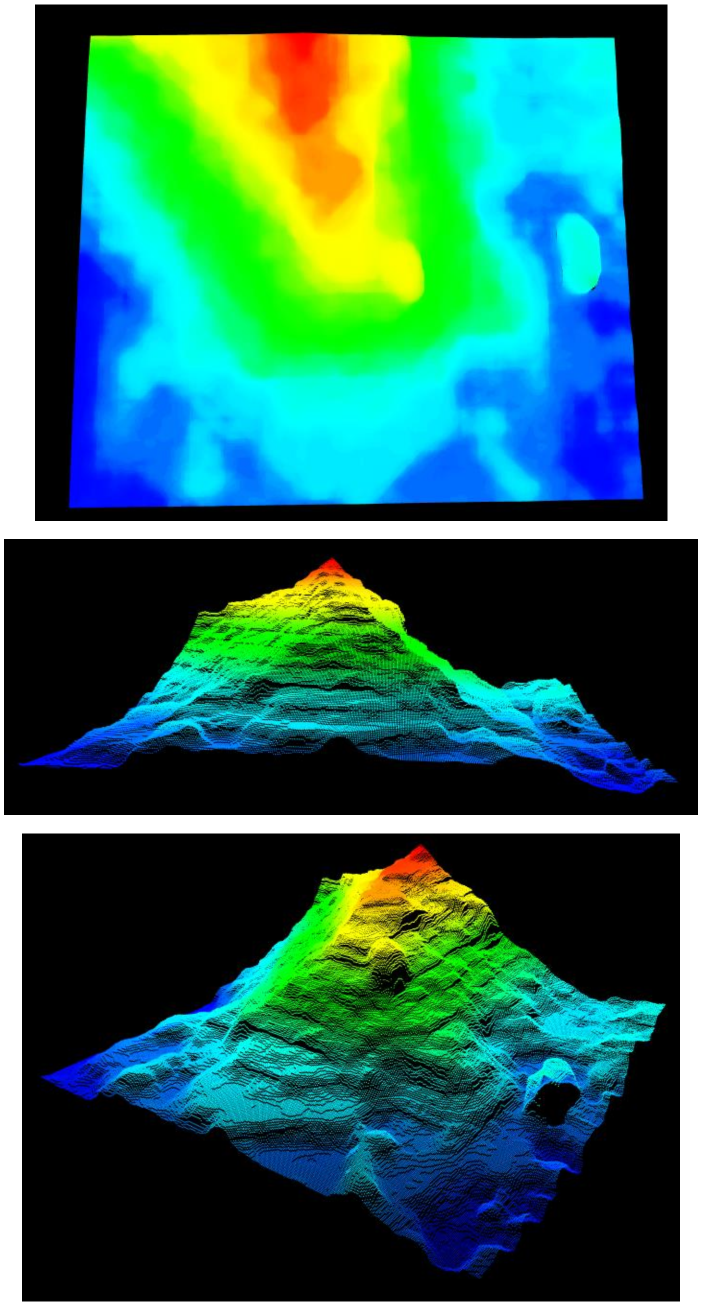


Figure 26. Point Clouds (test # 3)

4.2.2: Response of the Motors

The responses for 3 of the motors were plotted.

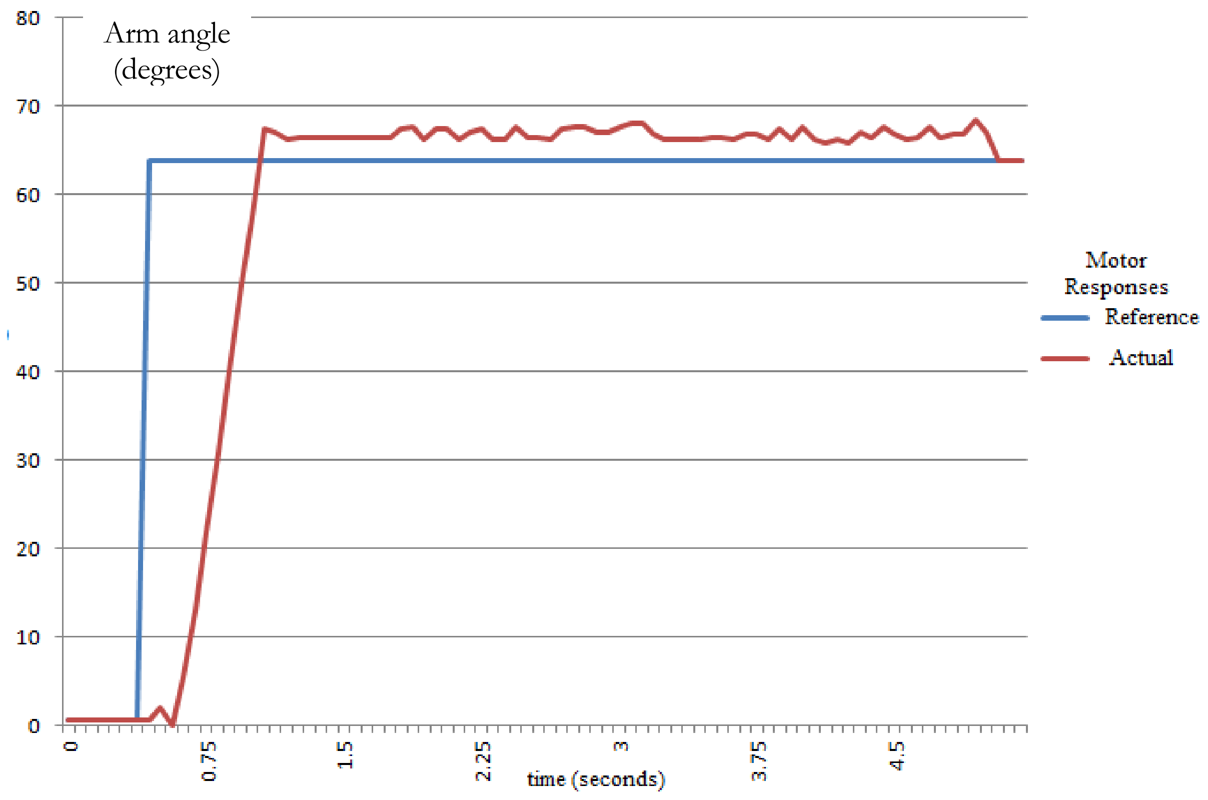


Figure 27. Arm angle response

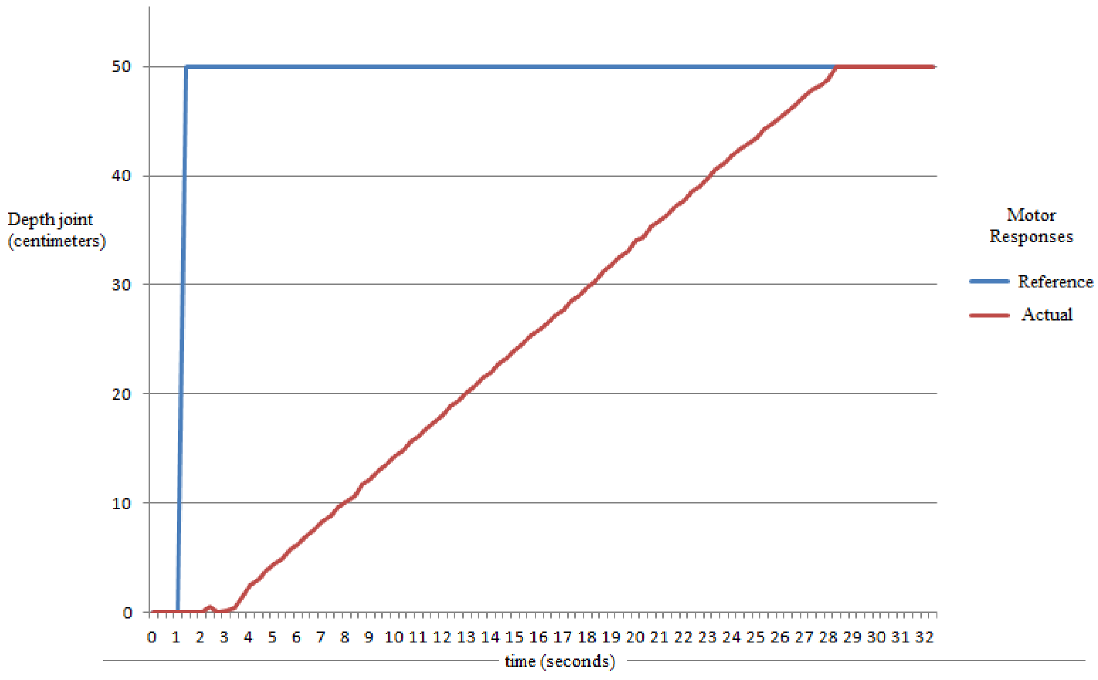


Figure 28. Depth motor response

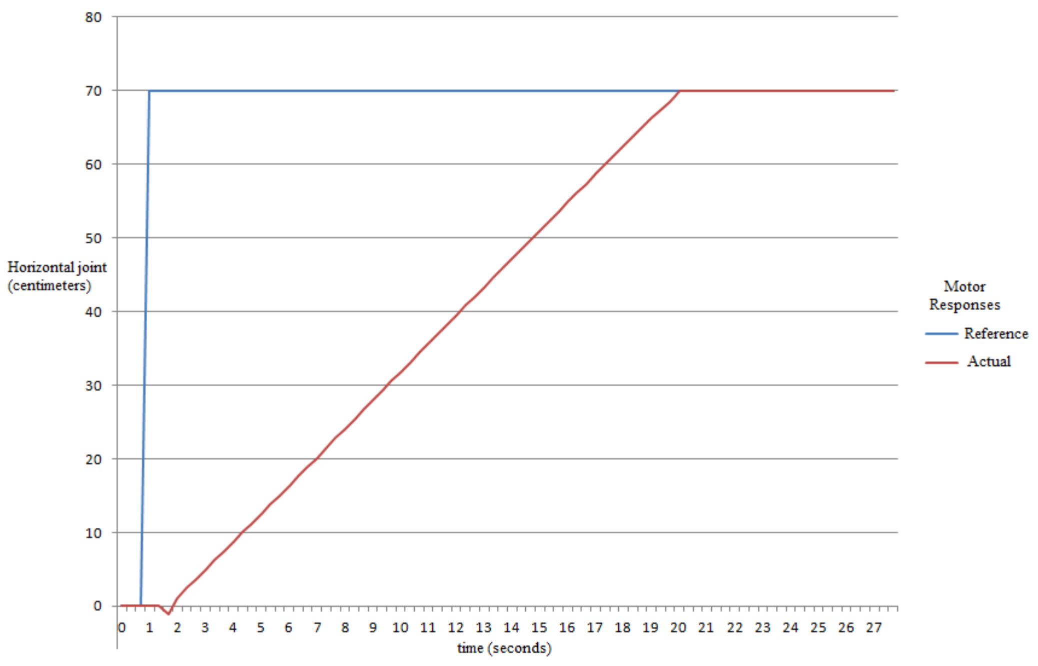
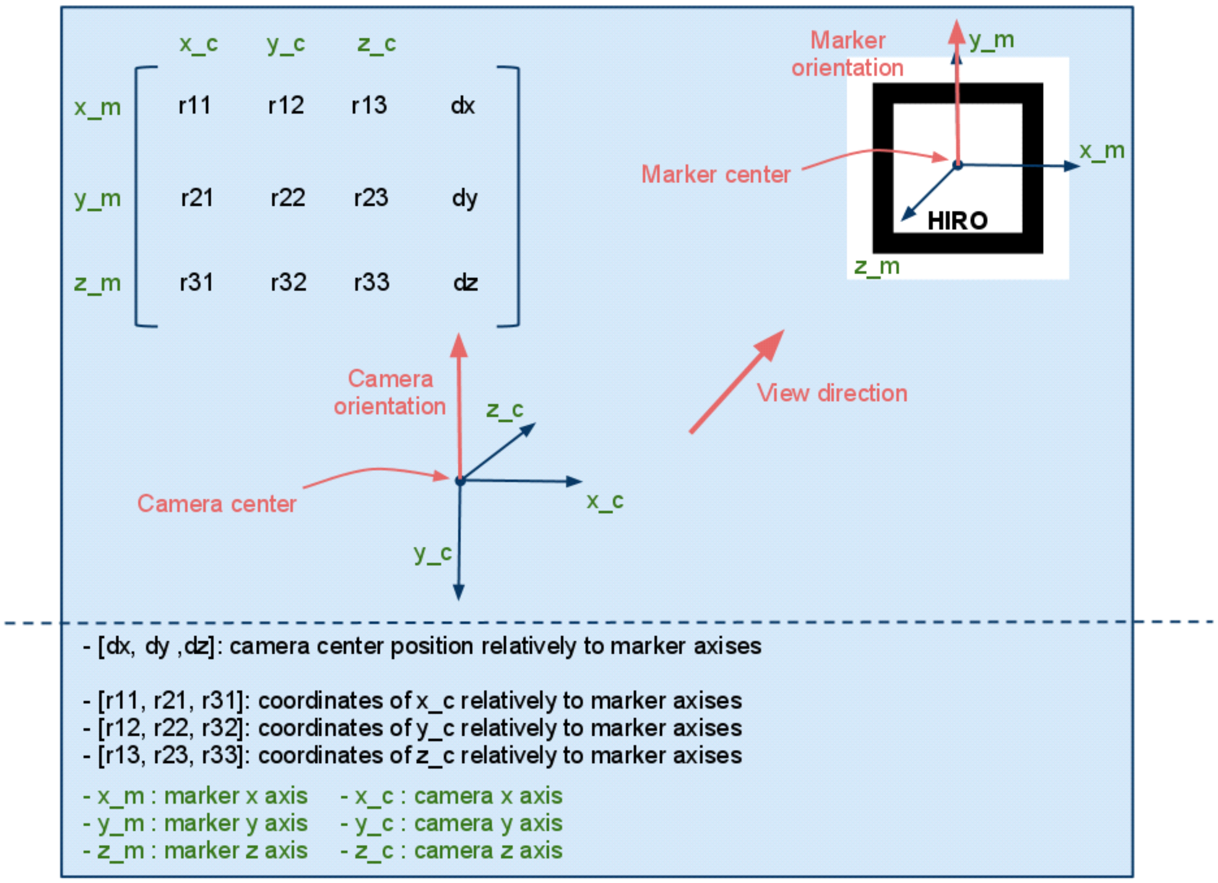


Figure 29. Horizontal motor response

4.2.3: 3D Pose Estimation Results

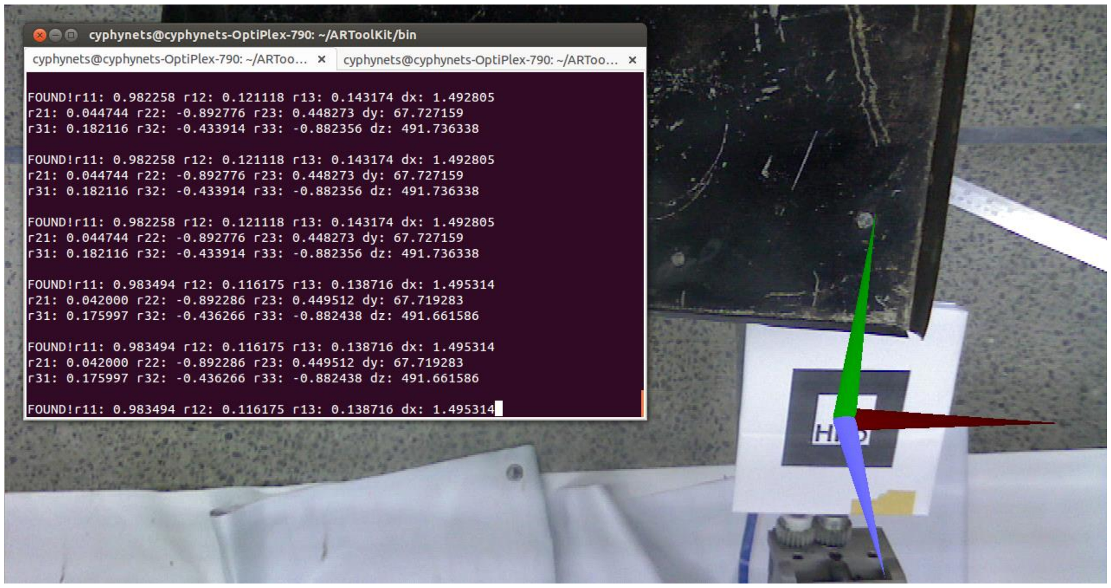
A screenshot of our 3D Pose Estimation run is shown in Figure 30. The axis of the detected end-effector is also visible. It lets us estimate the 3D pose using a function arGetTransMat. The result is a 3x4 rotation translation matrix Figure 30. The translation values are in millimeter, and the rotation value are normalized. One such result is as follows:



- [dx, dy ,dz]: camera center position relatively to marker axises

- [r11, r21, r31]: coordinates of x_c relatively to marker axises
- [r12, r22, r32]: coordinates of y_c relatively to marker axises
- [r13, r23, r33]: coordinates of z_c relatively to marker axises

- x_m : marker x axis	- x_c : camera x axis
- y_m : marker y axis	- y_c : camera y axis
- z_m : marker z axis	- z_c : camera z axis


cyphynets@cyphynets-OptiPlex-790: ~/ARToolKit/bin
cyphynets@cyphynets-OptiPlex-790: ~/ARToo...
cyphynets@cyphynets-OptiPlex-790: ~/ARToo...
HI

```
FOUND!r11: 0.982258 r12: 0.121118 r13: 0.143174 dx: 1.492805
r21: 0.044744 r22: -0.892776 r23: 0.448273 dy: 67.727159
r31: 0.182116 r32: -0.433914 r33: -0.882356 dz: 491.736338

FOUND!r11: 0.982258 r12: 0.121118 r13: 0.143174 dx: 1.492805
r21: 0.044744 r22: -0.892776 r23: 0.448273 dy: 67.727159
r31: 0.182116 r32: -0.433914 r33: -0.882356 dz: 491.736338

FOUND!r11: 0.982258 r12: 0.121118 r13: 0.143174 dx: 1.492805
r21: 0.044744 r22: -0.892776 r23: 0.448273 dy: 67.727159
r31: 0.182116 r32: -0.433914 r33: -0.882356 dz: 491.736338

FOUND!r11: 0.983494 r12: 0.116175 r13: 0.138716 dx: 1.495314
r21: 0.042000 r22: -0.892286 r23: 0.449512 dy: 67.719283
r31: 0.175997 r32: -0.436266 r33: -0.882438 dz: 491.661586

FOUND!r11: 0.983494 r12: 0.116175 r13: 0.138716 dx: 1.495314
r21: 0.042000 r22: -0.892286 r23: 0.449512 dy: 67.719283
r31: 0.175997 r32: -0.436266 r33: -0.882438 dz: 491.661586

FOUND!r11: 0.983494 r12: 0.116175 r13: 0.138716 dx: 1.495314
```

Figure 30. ARToolKit providing the visual feedback
(end-effector pose)

4.3: LIMITATIONS

4.3.1: Motor Play

The hardware setup has its own limitations. The play in the arm during angular movements does not allow the end effector to reach a desired place. Even during the horizontal movement, the arm keeps moving to and fro about its vertical axis and this doesn’t help in practical scenarios. The hardware setup needs to be modified so that the motor play is reduced or completely eliminated. In extreme cases, if the motor play isn’t reduced, the arm might hit obstacles/mines even if the motion planning module plans the trajectory perfectly. This can result in disastrous circumstances.

4.3.2: Physical Limitations of the Arm

Similarly, the arm has its own physical limitations. The motors can only take the arm to values within specific ranges. Taking the arm to extreme values and making the arm reach the physical limits results in erratic movements. This is often caused because of the fact that the motors require excessive amounts of current for moving in the physical limits of the setup.

4.3.3: Limitations of the Stereo Pair

The stereo pair sometimes fails to detect the end effector in the ROI and it results in jerky movements in such a way that the motion plan done previously is not followed at all. This can result in hitting the obstacles/mines as well. In turn, the code loses track of the end effector as well and can make erroneous plans and trajectories to be followed.

4.3.4: Non-linearity of the Mechanical Sensors

The sensors also return a range of values instead of a single value at a given point. This results in an error accumulation in the motion planning module and can cause serious failures. All the sensors exhibit somewhat linear behavior except the horizontal motor sensor. There is prominent nonlinearity in the behavior of the horizontal sensor and it results in faulty path planning by the motion planning module. The graphs below show the behavior of all the sensors.

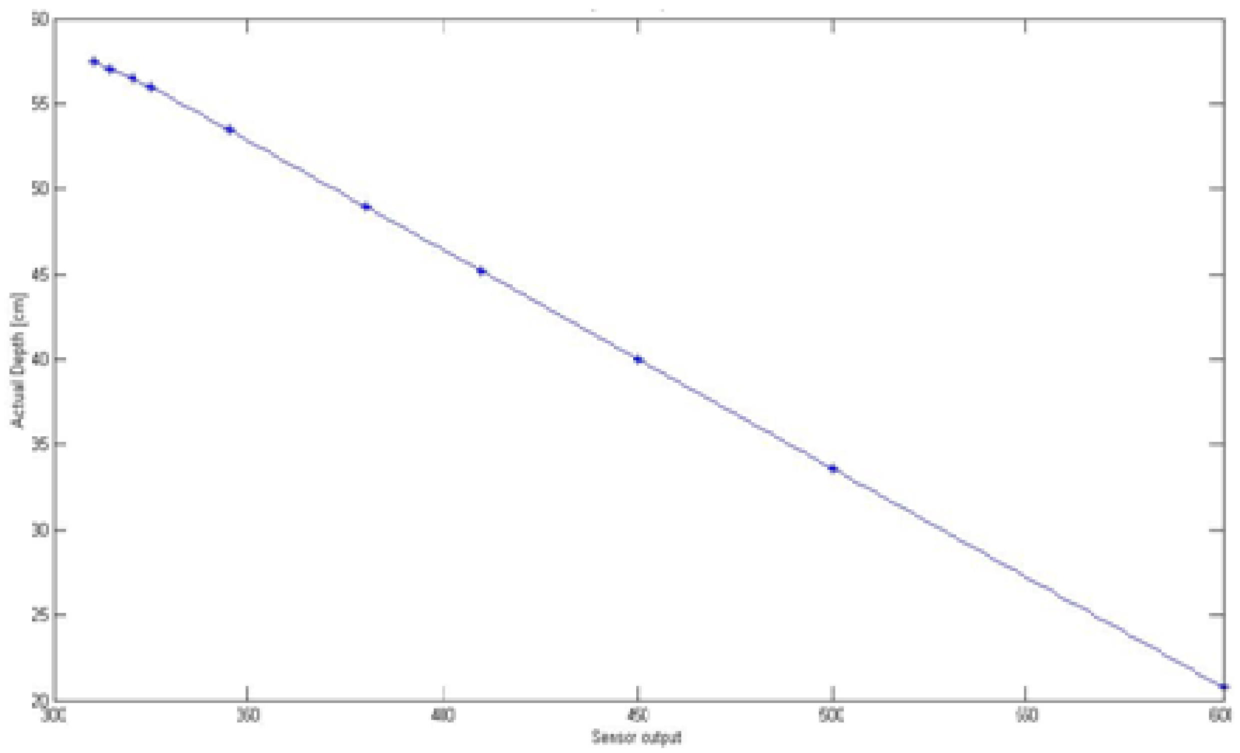


Figure 31. Depth Sensor

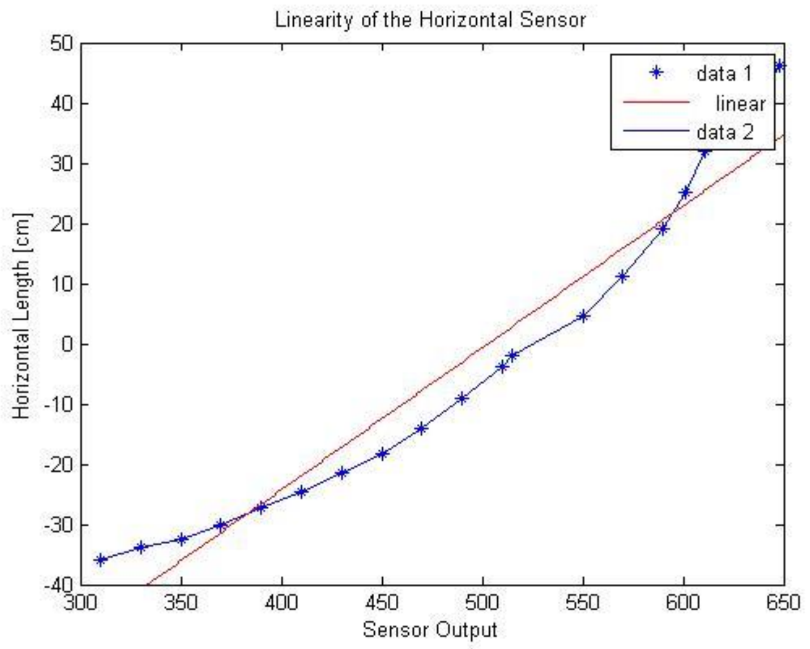


Figure 32. Horizontal Sensor

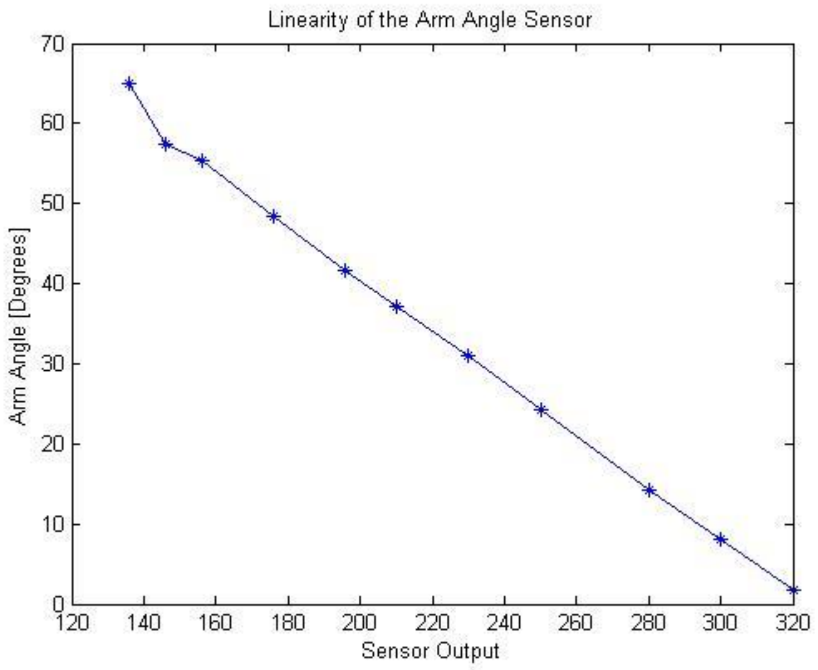


Figure 33. Arm angle Sensor

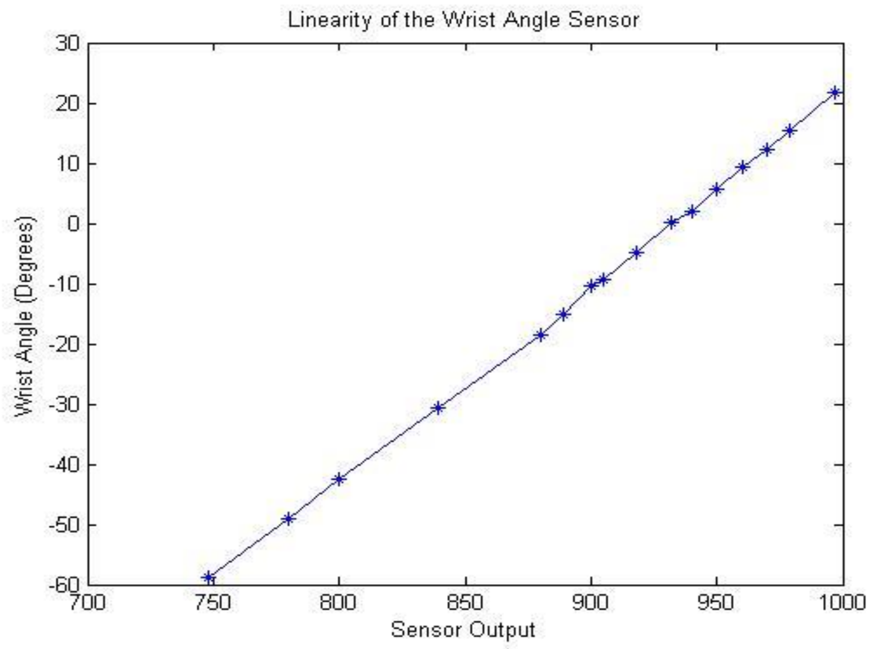


Figure 34. Wrist angle sensor

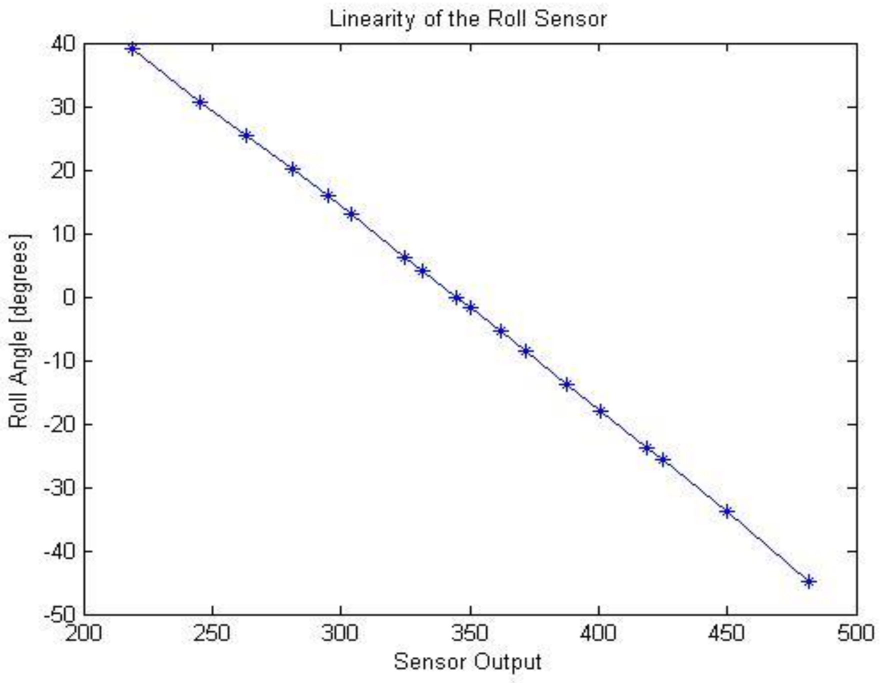


Figure 35. Wrist roll sensor

Chapter 5

COST ANALYSIS

Through this project, we have demonstrated that it is possible to construct a manipulator platform that can accurately track a terrain profile very closely and thus maintain a metal detector at an optimal distance from the landform to be scanned. The cost incurred in making this testbed running for lab experiments is shown in the Table 2.

Table 2. Cost Incurred

Item	**Cost (USD)**
Arduino MEGA 2560 Microcontroller	$24
Mechanical Sensors	$120
USB Stereo Cameras	$100
Metal Detector	$150
Power Supply (30 V / 10 A)	$280
Steel frame assembly	$250
Motor Drives and Power Electronics	$300
Total	$1224

Chapter 6

CONCLUSION AND FUTURE RECOMMENDATIONS

6.1: SENSOR FUSION

Sensor fusion is one task that needs to be done. We need to fuse the data coming from the two types of the sensors (stereo pair and potentiometers). Both make us informed about the current position of the end effector. Due to the nonlinearity of the horizontal sensors, it can return erroneous values. Field deployed mine detector arm will face bright sunlight which can result in malfunctioning of the stereo pair. Thus, sensor fusion can result in accurate description of the current position of the end effector in most of the situations. It can also allow us to rely totally on any one of the sensors depending upon the type of the environment.

6.2: LIVE VISUAL FEEDBACK FOR OBSTACLE AVOIDANCE

Obstacle avoidance on runtime will also be something for the future work. Currently, a motion plan for the arm is made offline and is executed on runtime as it is. We need to incorporate live visual / sensory feedback to avoid any dynamic obstacles even during the execution of the code. The motion planning module needs to run online for this purpose. The stereo pair can be made to take images of the terrain after specific intervals of time during the execution to make the motion planning module able to make an informed plan. The plan can be altered on runtime as well and the arm can circumvent any obstacles in its path to reach the desired location.

REFERENCES

[1] Bradski, G., Kaehler A.: Learning OpenCV: Computer vision with the OpenCV library. O'Reilly Media, 2008.

[2] Hirschmüller, H.: Stereo processing by semi-global matching and mutual information. IEEE Transactions on Pattern Analysis and Machine Intelligence 30(2): 328-341, 2008.

[3] Rusu, R.B., Cousins, S.: 3D is here: Point Cloud Library (PCL). IEEE International Conference on Robotics and Automation (ICRA). Shanghai, 2011.

[4] Martin, A.F., Robert, C.B.: Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Comm. of the ACM 24(6): 381–395, 1981.

[5] "ARToolKit Home Page." ARToolKit Home Page. N.p., n.d. Web. <http://www.hitl.washington.edu/artoolkit/>.

[6] "OpenCV | OpenCV." OpenCV | OpenCV. N.p., n.d. Web. <http://opencv.org/>.

[7] Muhammad, A., Abbas, S. M., Manzoor, T., Munawar, A., Abbas, S. A., Hayat, M., Abbas, A., Awais, M. M.: Marwa: A Rough Terrain Landmine Detection Robot For Low Budgets. Field and Assistive Robotics: Advances in Systems and Algorithms. Shaker Verlag, 2014.