IOWA STATE UNIVERSITY

# Developing Fast Multi-Object Tracking System for Hypervelocity Asteroid Intercept Vehicle

## Final Report

**Dec14-12**

Client : Dr. Bong Wie
Advisor : Dr. Meng Lu
Leader : Sean Nichols
Communications : Chi Hoe How
Webmaster : Yishu Mei

**12/8/2014**

## Contents

Introduction
Problem Statement
Concept Sketch
Experiment Setup
Implementation Details
System Block Diagram
System Description
User Interface Description
Functional Requirements
Non-Functional Requirements
Testing
Appendix

## Introduction

Our team was tasked to characterize an infrared sensor that is viable for the Hypervelocity Asteroid Interceptor Vehicle (HAIV). With the high cost of infrared sensing devices and our limited budget, we came up with a more feasible solution, which is to implement a detection and tracking system utilizing an optical camera.

The goal for this tracking system is to be able to process an image and update motor speed at a rate comparable to the frame rate of the camera input. With this real-time constraint, the tracking system was implemented with a combination of software with major hardware accelerators. The hardware accelerators implemented in this system are K-Means Clustering algorithm and stepper motor acceleration control. This way, the software can remain small and quick and solely focus on object tracking and error detection/handling.

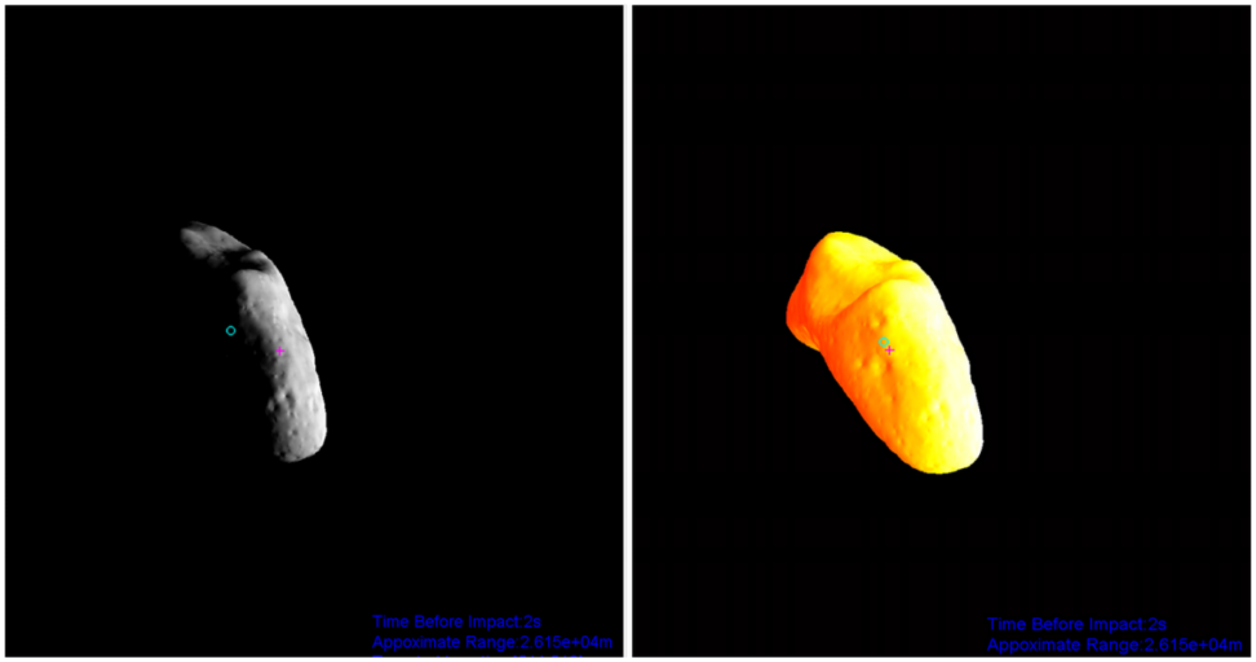

Figure 1. Comparison of optical image (left) and infrared image (right)

## Problem Statement

Our client and his team have been able to generate infrared and visible spectrum images of an asteroid and were able to simulate what images would look like if the asteroid was a great distance away. The parameters for their simulations are all theoretical and they desire to have more real-world parameters. They have also generated comparison images between the visible spectrum and the infrared spectrum. The problem is that they do not have any proof of how their simulated images or theoretical parameters compare to real values; we intend to provide this proof of concept and keep it very well documented so that our work can become of later use to them.

The feasibility of such a detection system was unknown to the client as there are many different infrared technologies available today that cover various ranges of the infrared spectrum as well as covering various ranges of operating temperature. The employment of these infrared optical lenses and mirrors was also unknown to the client. The distance of detection can be thought of as a distance between the sensor and the asteroid in space. This parameter was also an unknown quantity and clarification was sought by the client.

## Concept Sketch

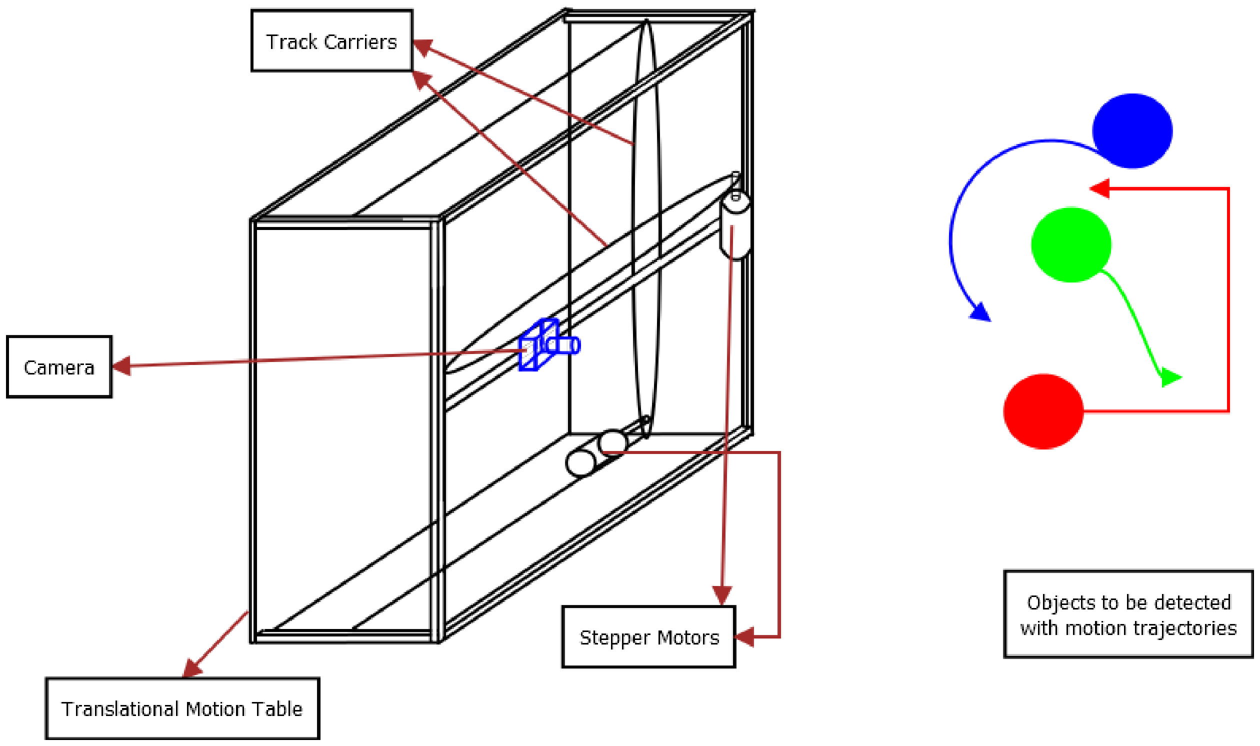
Track Carriers
Camera
Stepper Motors
Translational Motion Table
Objects to be detected with motion trajectories

Figure 2. Concept of asteroid detection and tracking system

## Experiment Setup

We plan to implement our system as a camera module placed on a translational motion table. This will be able to best emulate the action of thrusters in space (only in 1 plane of course). It will either be this, or on a pair of servos. The camera module will feed the digital system with video, the system will then process the incoming data and produce position change outputs to correct the position of the camera to keep the object of interest in the center of the camera's field of view.

Depending on the time available, we will strive to add other necessary features that could be used for HAIV as well. One such feature that we could add will be what we studied in our first semester of senior design : distance estimation. We determined distance between tracker and target from an image by knowing the optical system setup, camera pixel pitch, and known size of the object. From this, we were able to generate a rough mathematical relationship between number of pixels to distance of object. If we complete our primary objective with extra time left over, then we will strive to implement this.

For our experiment, the object of interest will be a small glowing ball to mimic black body radiation in the infrared spectrum. Since we will be dealing within the visible range, we will need to block out all other light sources to properly emulate the environment of space for our experiment. The camera as well as the object of interest will simply be placed in a dark box. That way, only the object of interest, as well as some smaller "radiating" objects will be placed in the box to show that our system works even with a small amount of "noise."

## Implementation Details

### Hardware

Our project is split into two categories the hardware and software side of things. Our image processing pipeline is done in hardware for a faster processing time since we do not want delays when detecting and tracking our objects. For the object detection algorithm, we implemented K-means Clustering. K-means Clustering analyzes input pixels of the object to be detected. Then with set centroids randomly distributed on the image, the distance is calculated from each centroid to each individual pixel value. Then the minimum distances from every pixel to the centroid is taken and averaged together to place the centroid in the middle of the cluster of pixels. These clusters are then grouped together as an object but if the centroid is not in the center of the cluster, multiple passes are done again to average and group the distance data.

The K-means Clustering hardware core is implemented to be fully configurable via register sets and sends data using the Xilinx AXI4-Stream video protocol interface. It also has 16 object detection cores to detect up to 16 objects simultaneously with zero image processing pipeline latency. After the centroids are placed in the detected center of the object, pixel "painting" is done through this core to easily identify the objects for software processing. This hardware core also counts the row and column to determine the position of the object and it returns these values of row and column of detected objects. K-Means Clustering core mostly consists of grouper and averager modules. The grouper performs distance calculation with 16 cores and determines the minimum distance of a single centroid point with all the pixels on screen. When there is a match, the grouper tags the match with the object ID and the averager accumulates all the matched ID minimum distances and divides it with the number of matched pixels to find a new mean for the centroid.

The figures below summarizes the K-means Clustering process where the random centroid points are initially placed and then it starts grouping pixel clusters through averages of minimum distances calculated. Then the location is updated and the centroids move closer its designated pixel cluster.

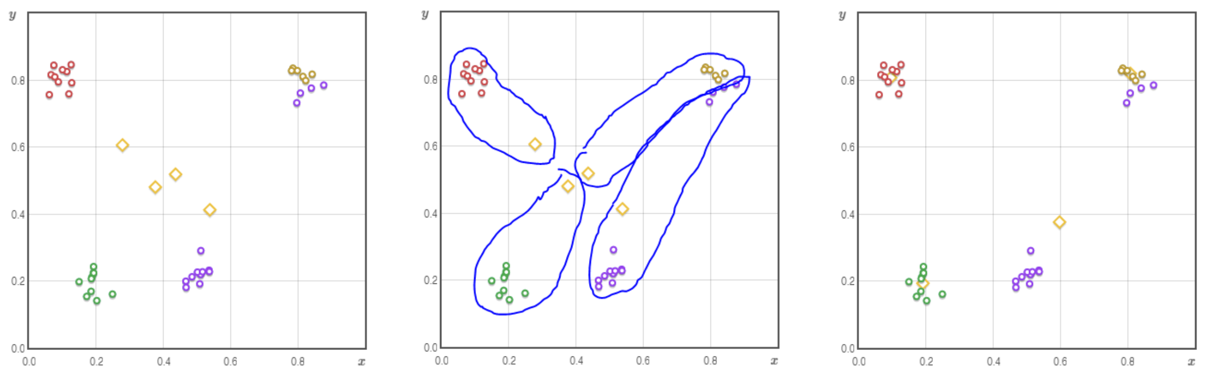

Figure 3. Initialization, Grouping, and Update of mean values

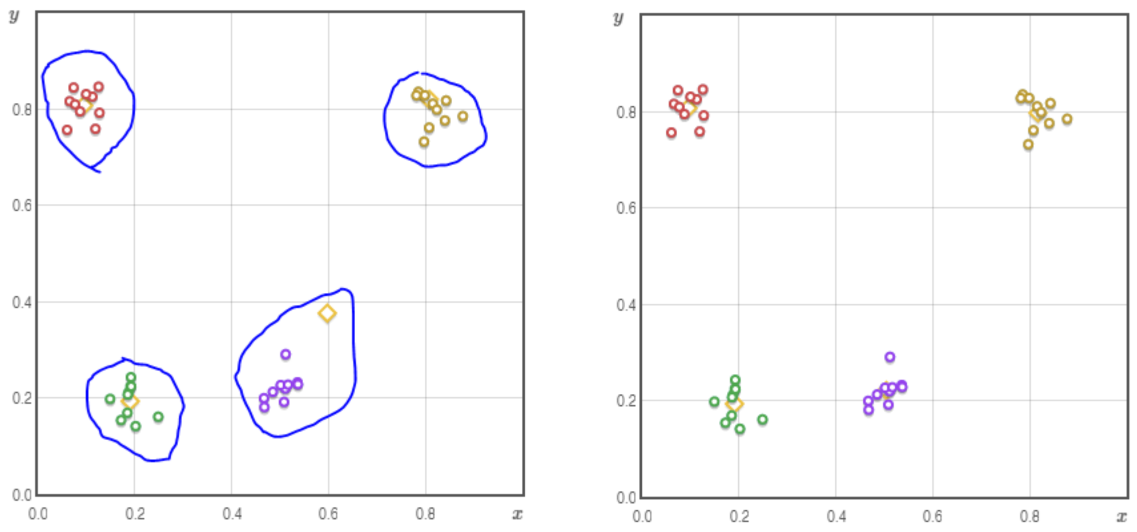

Figure 4. Second pass of k-means algorithm

The camera used in the tracking system is moved via a stepper motor which we also control using hardware modules. The module implemented controls the acceleration and deceleration of the stepper motor. Similar to the K-means Clustering unit, it is fully configurable via register sets and consists mainly of dividers and counters. Besides that, the stepper motor core also features automatic handling of the "direction" bit and stopping of motor at limit boundaries to avoid breaking of the motors.

## Software

On the software side, the object tracking is handle here with no image processing as it is done in hardware. The software is made to allow us to choose a single object to track and follow even when multiple objects are present. Implemented on software is the ability to characterize parameters of the tracked object such as the relative speed, current position, last know position and also its future position with our prediction algorithm shown below.

We also implemented a P-D controller that uses error correction to smoothly center the camera over the object of interest. With this the object of interest should always be displayed in the center of the screen displaying the live video feed.

Working in-line with the P-D controller, we have an object tracking prediction algorithm implemented that uses previous known positions of the object and adds a vectored distance based on sample time and relative object speed. These values can then determine the trajectory of the single object in a sea of multiple objects present.

$$p_{pred} = p_{old} + (speed * time)$$

Equation 1. Object trajectory prediction equation

## System Block Diagram

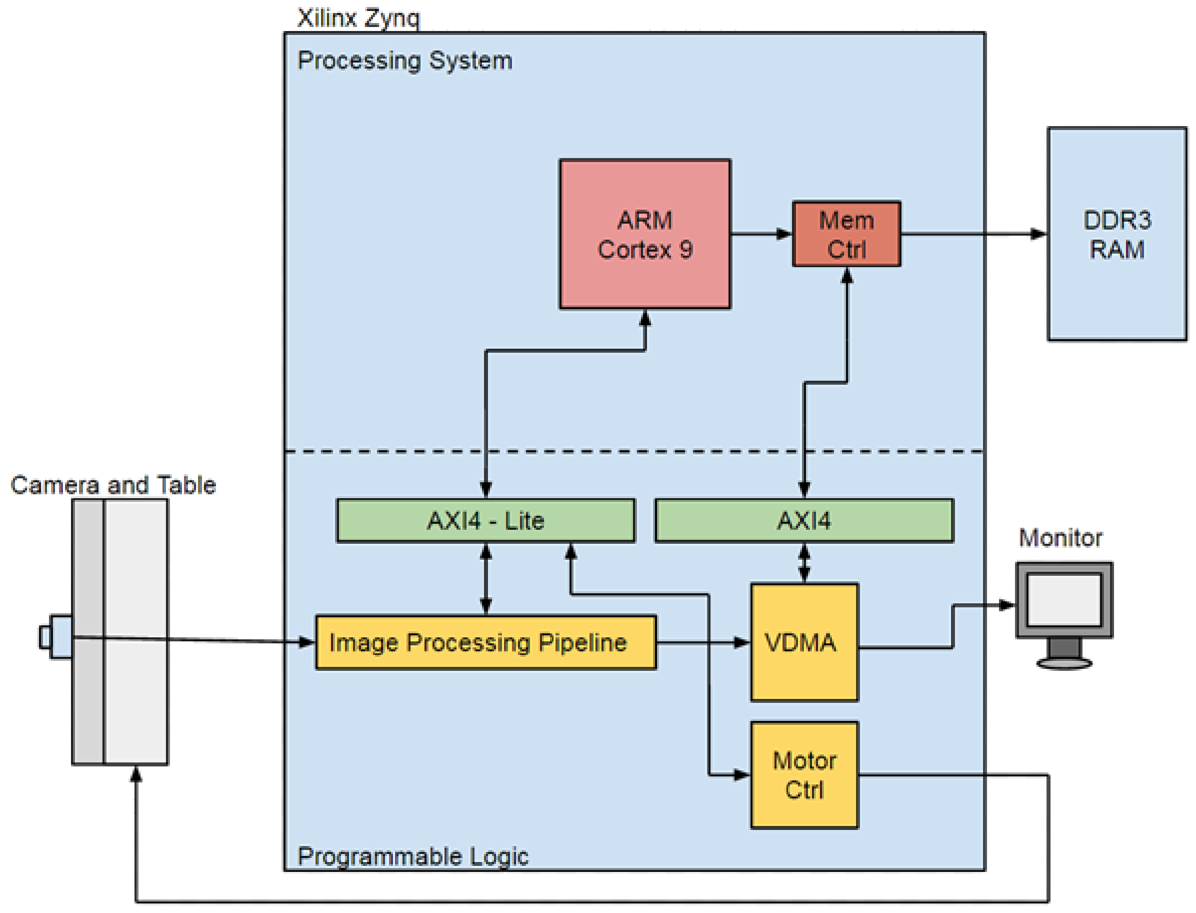

Figure 5. Top level system block diagram

Explanation

## System Description

The camera module by default outputs YCbCr data format, so we need to setup the hardware to handle this. The video input module will handle taking an image from the camera and turning it into a pixel stream. Once we have this stream, we will run the image through a processing pipeline in hardware. This will offload the processor since that will be a very "slow" operation in software, but relatively "fast" in hardware. This will help keep the framerate high. Once the image has passed the video processing pipeline, the processed image will then be pushed into main memory via the Video Direct Memory Access (VDMA) engine. It will be here that software will be able to view the processed image (most likely sobel filtered) and perform quick processing on it to determine what direction to correct the camera. The processor will then output the proper controls to the motors as well as the new video data to a monitor so users can see what the system is seeing.

## User Interface Description

Since the ZedBoard has a serial communication interface, we will be using PuTTy to construct a simple GUI which allows us to set some options and send back important data and debug messages.

## Functional Requirements

The functional requirements for our tracking system is the ability to detect multiple objects. Do note that detect and track has different meanings and detecting means knowing there are objects in the camera's field of view. The requirement to be able to track an object at high speed is very crucial because an asteroid travels at great speeds.

Also, the time taken for the image processing has to occur under 16ms, the frame period, which is derived from the desired frame rate of 60fps. With multiple objects detected to be in the field of view, we also need the ability to choose which object to track as not every object can be followed with only a single camera available.

## Non-Functional Requirements

These non-functional requirements allow the system to work more smoothly but is not needed to have the system running. Video feedback is considered a non-functional requirement because the system can still run without live video feed coming from the camera. As long as the image taken by the camera is being processed and output data is being sent to the motor controls, the system should still be running correctly.

Pixel "painting" is the process of coloring the objects detected on the input image taken from the camera and displaying it on the video feedback. Since it is a part of the video feedback, it is also considered to be a non-functional requirement.

## Testing

Testing of this system imitates the conditions in space where multiple colored ping pong balls acting as asteroids are placed arbitrarily in a black box or has a black cloth as a background. The ping pong balls will emit certain colors (wavelengths of visible light) with LEDS, to simulate a black body radiator. We will then move the lighted balls to show the tracking system with the camera relative to the speed of the object.

<insert picture of translational motion table here>

# Appendix

## Appendix I: AXI4-Stream K-Means Clustering Core Register Definition

**Register 0x00 - Status**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Total Match Count | | | | | | | | | | | | | | | | Reserved | | | | | | | | | | | | | | Done | Busy |

**Note :** This register is RO Busy bit is set and cleared by hardware. Done bit is set by hardware and cleared by a write to the Start bit in the control register.

**Register 0x01 - Control**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Object Enable Field | | | | | | | | | | | | | | | | Reserved | | | | | | | | | | | Paint En | Num Pass | | Mult Obj | Start |

**Note :** This register is R/W. A write to the start bit clears the Done bit. “Paint En” enables painting of matched pixels. “Num Pass” determines number of passes of K-means core. “Mult Obj” tells core that multiple objects are present. More than one object enable must be set!

**Register 0x02 – Upper Pixel Threshold**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | Red Value | | | | | | | | Green Value | | | | | | | | Blue Value | | | | | | | |

**Note :** Pixel match max values. Will match when all components are less than or equal to the value in this register

**Register 0x03 – Lower Pixel Threshold**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | Red Value | | | | | | | | Green Value | | | | | | | | Blue Value | | | | | | | |

**Note :** Pixel match min values. Will match when all components are greater than or equal to the value in this register

**Register 0x04 – Pixel Match Paint Value**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | Red Value | | | | | | | | Green Value | | | | | | | | Blue Value | | | | | | | |

**Note :** If the "Paint En" bit is set in the control register, then this will determine the color that replaces the current matched pixel color.

**Register 0x05**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | Object 0 Mean Row | | | | | | | | | | | Object 0 Mean Col | | | | | | | | | | |

**Note :** This register contains the centroid of the object 0. All subsequent registers of this type are similar to this. May be writable by software in a future version, but for now all initial values are hardcoded.

**Register 0x06**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | Object 1 Mean Row | | | | | | | | | | | Object 1 Mean Col | | | | | | | | | | |

**Register 0x07**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | Object 2 Mean Row | | | | | | | | | | | Object 2 Mean Col | | | | | | | | | | |

**Register 0x08**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | Object 3 Mean Row | | | | | | | | | | | Object 3 Mean Col | | | | | | | | | | |

Register 0x09

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | Object 4 Mean Row | | | | | | | | | | | Object 4 Mean Col | | | | | | | | | | |

Register 0x0A

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | Object 5 Mean Row | | | | | | | | | | | Object 5 Mean Col | | | | | | | | | | |

Register 0x0B

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | Object 6 Mean Row | | | | | | | | | | | Object 6 Mean Col | | | | | | | | | | |

Register 0x0C

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | Object 7 Mean Row | | | | | | | | | | | Object 7 Mean Col | | | | | | | | | | |

Register 0x0D

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | Object 8 Mean Row | | | | | | | | | | | Object 8 Mean Col | | | | | | | | | | |

Register 0x0E

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | Object 9 Mean Row | | | | | | | | | | | Object 9 Mean Col | | | | | | | | | | |

**Dec14-12**

**Register 0x0F**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | Object 10 Mean Row | | | | | | | | | | | Object 10 Mean Col | | | | | | | | | | |

**Register 0x10**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | Object 11 Mean Row | | | | | | | | | | | Object 11 Mean Col | | | | | | | | | | |

**Register 0x11**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | Object 12 Mean Row | | | | | | | | | | | Object 12 Mean Col | | | | | | | | | | |

**Register 0x12**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | Object 13 Mean Row | | | | | | | | | | | Object 13 Mean Col | | | | | | | | | | |

**Register 0x13**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | Object 14 Mean Row | | | | | | | | | | | Object 14 Mean Col | | | | | | | | | | |

**Register 0x14**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | Object 15 Mean Row | | | | | | | | | | | Object 15 Mean Col | | | | | | | | | | |

Dec14-12

**Register 0x15**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Object 0 Match Count | | | | | | | | | | | | | | | | Object 1 Match Count | | | | | | | | | | | | | | | |

**Note :** This register contains the total match count for an object.

**Register 0x16**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Object 2 Match Count | | | | | | | | | | | | | | | | Object 3 Match Count | | | | | | | | | | | | | | | |

**Register 0x17**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Object 4 Match Count | | | | | | | | | | | | | | | | Object 5 Match Count | | | | | | | | | | | | | | | |

**Register 0x18**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Object 6 Match Count | | | | | | | | | | | | | | | | Object 7 Match Count | | | | | | | | | | | | | | | |

**Register 0x19**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Object 8 Match Count | | | | | | | | | | | | | | | | Object 9 Match Count | | | | | | | | | | | | | | | |

**Register 0x1A**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Object 10 Match Count | | | | | | | | | | | | | | | | Object 11 Match Count | | | | | | | | | | | | | | | |

**Dec14-12**

Register 0x1B

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Object 12 Match Count | | | | | | | | | | | | | | | | Object 13 Match Count | | | | | | | | | | | | | | | |

Register 0x1C

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Object 14 Match Count | | | | | | | | | | | | | | | | Object 15 Match Count | | | | | | | | | | | | | | | |

## Appendix II: Stepper Motor Control Core Register Definition

**Register 0x00 – Interface (status and control)**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | | | Enable | Sw Active State | Max Err | Min Err | Clr Step Cnt |

Max/Min Err are high when the motor has been killed and a boundary switch has been hit. Sw active state tells hardware if the switches are active high or active low. Clear step cnt clears the step count. If a boundary fault occurs, then the core will be useless until the error has been cleared. Max Err and Min Err are write to clear (write a 1 to this bit to clear). The Enable bit must also be high, in order for the core to operate. The motor can be killed at anytime by clearing the enable bit

**Register 0x01 – Acceleration Increment Period (ns)**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Acceleration Increment Period (ns) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

This register contains information about acceleration time unit in nanoseconds

**Register 0x02 – Acceleration value**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Acceleration Value (ns) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

This register contains the desired acceleration increment value. This is the value that is either added or subtracted from the current speed periodically as described by the acceleration increment period

**Register 0x03 - Step period ns**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Dir | Step Period (ns) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

This register takes in the desired period in nanoseconds of the step output. You can go forward or backward based on bit 31 of this register. Hardware automatically handles stopping and starting when changing directions

**Register 0x04 – Step count**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Step Count | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

This register can be cleared by software, but is also cleared when the direction is switched

**Register 0x05 – Current step period**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Current Step Period (ns) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

This register contains the current step clk period value in nanoseconds