Zebravision 3.5 – Team 900

Zebravision 3.5 is Team 900's work done in the postseason of 2015 and preseason of 2016. Our focus has been finding new ways to gather information about our robot's surroundings. We have been working to integrate the StereoLabs ZED, a stereoscopic camera, into the 2015 vision system with the Jetson TK1.

Evolution of Zebravision

Zebravision was first used during the 2012 season. Between then and 2014, the system used Axis 206 cameras. During 2012 and 2013 (Zebravision 1.0 and 2.0, respectively) processing was done on the driver station by transferring video over the network. After bandwidth restrictions imposed by FRC, we moved to an onboard computation model, originally using the cRIO to process data. The cRIO was only able to process single frames and was severely limited in terms of usefulness.

These issues prompted the creation of Zebravision 3.0, the 2014 solution to limits on vision processing. Our solution included using OpenCV, a vision library, in conjunction with the NVIDIA Jetson TK1 DevKit mounted on the robot. We also started using the Logitech C920, which improved framerate and video quality. The implementation of this system hugely increased our capabilities. With this increase in processing power, we are now able to develop more powerful vision processing methods. More information about this can be found in the Zebravision 3.0 whitepaper here.

The ZED Camera

Overview and Technical Specifications

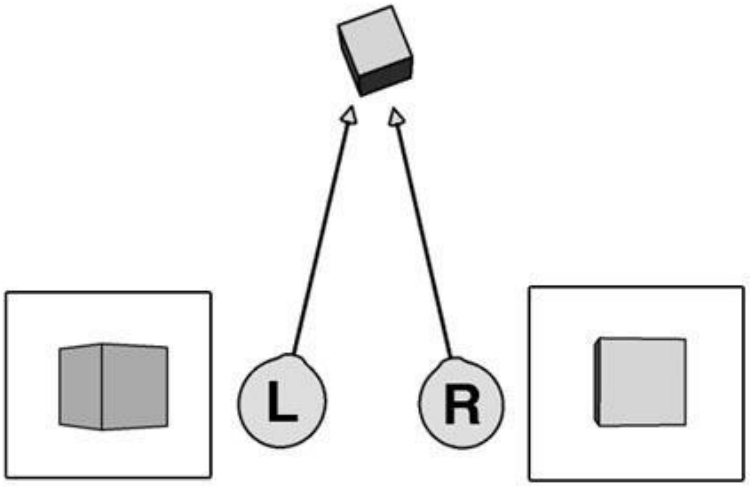
The ZED is a high definition 3D stereoscopic camera designed for real time depth sensing in robotics applications.

Each camera captures up to 2.2K resolution. The framerate maximum depends on the resolution requested. At 2.2K the camera can capture 15fps, but at VGA resolution (1280x480) it can capture 120fps. Because of technical limitations on the Jetson, the camera can only output at 720p and 15fps when using the Jetson. The depth stream is always output at the same resolution as the main camera.

Both the left and right streams are output simultaneously over a USB 3.0 port. The camera can capture depth from a range of 3.5 feet to 68 feet.

How it works

The ZED computes depth using a technique called disparity:



This technique involves comparing the left and right images and finding the difference between them. A smaller difference between the two frames indicates that the object is further away. This technique is commonly used in robotics and sometimes used in FRC (in fact it was used in Zebravision 2.0). The advantage of the ZED camera is that most disparity computation requires careful calibration; however, since the ZED has two cameras that cannot rotate independently it comes precalibrated, which allows greater accuracy out of the box. Additionally, the depth map is computed on the GPU with highly efficient CUDA code, which is much faster than OpenCV processing running on the CPU alone.

Using the ZED

Requirements

- We have only tested this process using Ubuntu 14.04 but the libraries run on Windows and other distributions of Linux as well.
- The computer must have a NVIDIA GPU. This is required for the ZED libraries. For use on robots we recommend using the NVIDIA Jetson TK1.
- The ZED camera.

Setup

Stereolabs has created the ZED SDK. This is the interface necessary for the ZED to get images into OpenCV, an open source vision library. The first step in the process is installing OpenCV. OpenCV comes preinstalled with the Jetson so if you are using it you do not need to worry about this step. If you are not using the Jetson, copy the script from this link, paste it into a blank text document named install_opencv.sh and run the following commands:

```
chmod a+x install_opencv.sh
```

```
./install_opencv.sh
```

After this runs you should have OpenCV installed. The next step is to install the CUDA toolkit, which enables support for building the ZED SDK. This is again not necessary if you're using the Jetson. You can find installation packages here.

The Jetson comes with USB 3.0 disabled by default. To use the ZED you have to enable it with this command:

```
sudo gedit /boot/extlinux/extlinux.conf
```

Next, change the line that says *usb_port_owner_info=0* to *usb_port_owner_info=2.* Restart, then move on to the next step.

Now it's time to install the ZED SDK. You can get the download from here. Make sure to download the Jetson version if you are on the Jetson. Run the following commands in your download location:

```
chmod a+x [name of the file].sh
```

```
./[name of the file]
```

Then follow the directions to install the SDK. Installing the samples when prompted is a good idea.

Using in code

The full documentation for the SDK can be found online here. To use the SDK in your program, include these lines in a cmake file linked to your program. If you are using our code this is done for you.

```
find_package(ZED 0.8.2 REQUIRED)
find_package(CUDA 6.5 REQUIRED)
include_directories(${ZED_INCLUDE_DIRS})
include_directories(${CUDA_INCLUDE_DIRS})
```

and add `${ZED_LIBRARIES} ${CUDA_LIBRARIES}` to your target_link_libraries() statement. Then add these lines to the top of your main C++ file:

```
//zed include
#include <zed/Mat.hpp>
#include <zed/Camera.hpp>
#include <zed/utils/GlobalDefine.hpp>

//cuda include
#include "cuda.h"
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include "npp.h"
#include "device_functions.h"
```

Now you can use functions to pull data from the ZED. The full list of functions can be found in the documentation online. Here is a basic example of how to pull and display ZED images using OpenCV:

```
1   sl::zed::Camera zed = new sl::zed::Camera(sl::zed::VGA);
2   zed->init(sl::zed::MODE::QUALITY, -1, true);
3   int width = zed->getImageSize().width;
4   int height = zed->getImageSize().height;
5   cv::Mat imageCPU(height, width, CV_8UC4);
6   cv::Mat depthCPU(height, width, CV_8UC4);
7
8   while(1) {
9       zed->grab(sl::zed::FULL);
10
11      sl::zed::Mat imageGPU = zed->getView_gpu(sl::zed::STEREO_LEFT);
12      cudaMemcpy2D((uchar*) imageCPU.data, imageCPU.step, (Npp8u*) imageGPU.data, imageGPU.step,
13      imageGPU.getWidthByte(), imageGPU.height, cudaMemcpyDeviceToHost);
14      cvtColor(imageCPU,imageCPU,CV_RGBA2RGB);
15
16      sl::zed::Mat depthGPU = zed->normalizeMeasure_gpu(sl::zed::MEASURE::DEPTH);
17      cudaMemcpy2D((uchar*) frame.data, frame.step, (Npp8u*) depthGPU.data, depthGPU.step,
18      depthGPU.getWidthByte(), depthGPU.height, cudaMemcpyDeviceToHost);
19      cvtColor(depthCPU,depthCPU,CV_RGBA2RGB);
20
21      imshow("Depth", depthCPU);
22      imshow("Left", imageCPU);
23      waitkey(5);
24  }
```

Lines 1 and 2 initialize the ZED. "sl::zed::VGA" is the quality setting for the ZED. Lines 4-5 create width and height variables that are used in the next two lines to create image containers that store the images from the ZED. The loop starting on line 8 runs for every frame. Lines 11-13 pull the left frame from the ZED, store it in a GPU image, and then copy that image from the GPU to an OpenCV image on the CPU. Line 14 is a conversion from the 4 channel image created by the ZED to the 3 channel image required by OpenCV's imshow() method.

Note that the depth map that is output is not the depth but rather a normalized depth with values between 0 and 255. To get the actual depth use the following code:

```
25      depthMat = zed->retrieveMeasure(sl::zed::MEASURE::DEPTH);
26      float* data = (float*) depthMat.data;
27      float* ptr_image_num = (float*) ((int8_t*) data + y * depthMat.step);
28      float dist = ptr_image_num[x] / 1000.f;
```

This code takes an X and Y value and finds the depth at that location in meters.

Lines 16-19 are the same as 11-14 except they process the depth map rather than the camera's frames. Lines 21-23 are for displaying the images. Line 23 is required for the imshow() method to work.

Zebravision 3.0 integration

We integrated the ZED into our bin detection and tracking code from last season. Last season we used trigonometry to determine the distance to the bin based on the size at the time of detection. We switched to using the ZED to determine distance. This has proved to be far more accurate with less variability between frames.

Looking Forward

The addition of depth to our vision data greatly increases the scope of our vision programming. We are planning to replace classifiers with neural networks this upcoming year. The addition of a fourth

dimension will increase the accuracy of classification by neural networks. Another possible application includes creating a map in real time by combining depth and positional data. Overall the ZED is a powerful piece of equipment for the future of vision in robotics.

Thanks

Special thanks to Stereolabs for providing support and assistance as we tested and integrated the ZED into our own work. We could not do this without their support!