COMPUTER VISION BASED ROBOT NAVIGATION

ABU ROBOCON 2016

ANMOL POPLI

NITESH ARORA

TEAM ROBOCON - IIT ROORKEE

Faculty Advisor

Prof. Swain Abinash Kumar

Department of Mechanical and Industrial Engineering

CONTENTS

INTRODUCTION

This work was done as a module for the problem statement of the robotics contest ABU ROBOCON 2016. The outline of the contest was as follows. Each team consists of two robots: one Eco Robot and one Hybrid Robot. Eco Robot doesn't have an actuator to drive itself. The driving force of Eco Robot is obtained indirectly from Hybrid Robot; for example, wind force, magnetic force, etc., or from the game field structure, gravity force, etc. Eco Robot carrying Wind Turbine Propeller departs from "Eco Robot Start Zone". It runs along 3 zones; "3 Slopes and Hills" "River" "Down Hill", and aims for "Wind Turbine Station" by receiving driving energy from Hybrid Robot. Eco Robot has to use only one steering actuator to control its direction, to track the path containing Slopes and Hills, River, and Down Hill. After Eco Robot reaches at "Wind Turbine Station", Hybrid Robot gets Wind Turbine Propeller from Eco Robot. Then Hybrid Robot climbs Wind Turbine Pole and assembles Wind Turbine Propeller on Wind Turbine Engine attached on top of Wind Turbine Pole. The team that successfully assembles Wind Turbine Propeller earlier is the winner of the game.

This module comprised the autonomous steering of the Eco Robot along the path specified, employing Computer Vision. The non-contact force we used for driving was wind force by propellers mounted on the Hybrid Bot. A DC Servo Motor was used as the steering actuator in the Eco Robot. The Camera on an Android Phone was used for Vision feedback from the arena. We developed an Android application for processing the images captured by the camera. Post processing, the Android device communicates the command angle to the Arduino microcontroller via USB. Arduino controls the angular position of the servo motor accordingly.

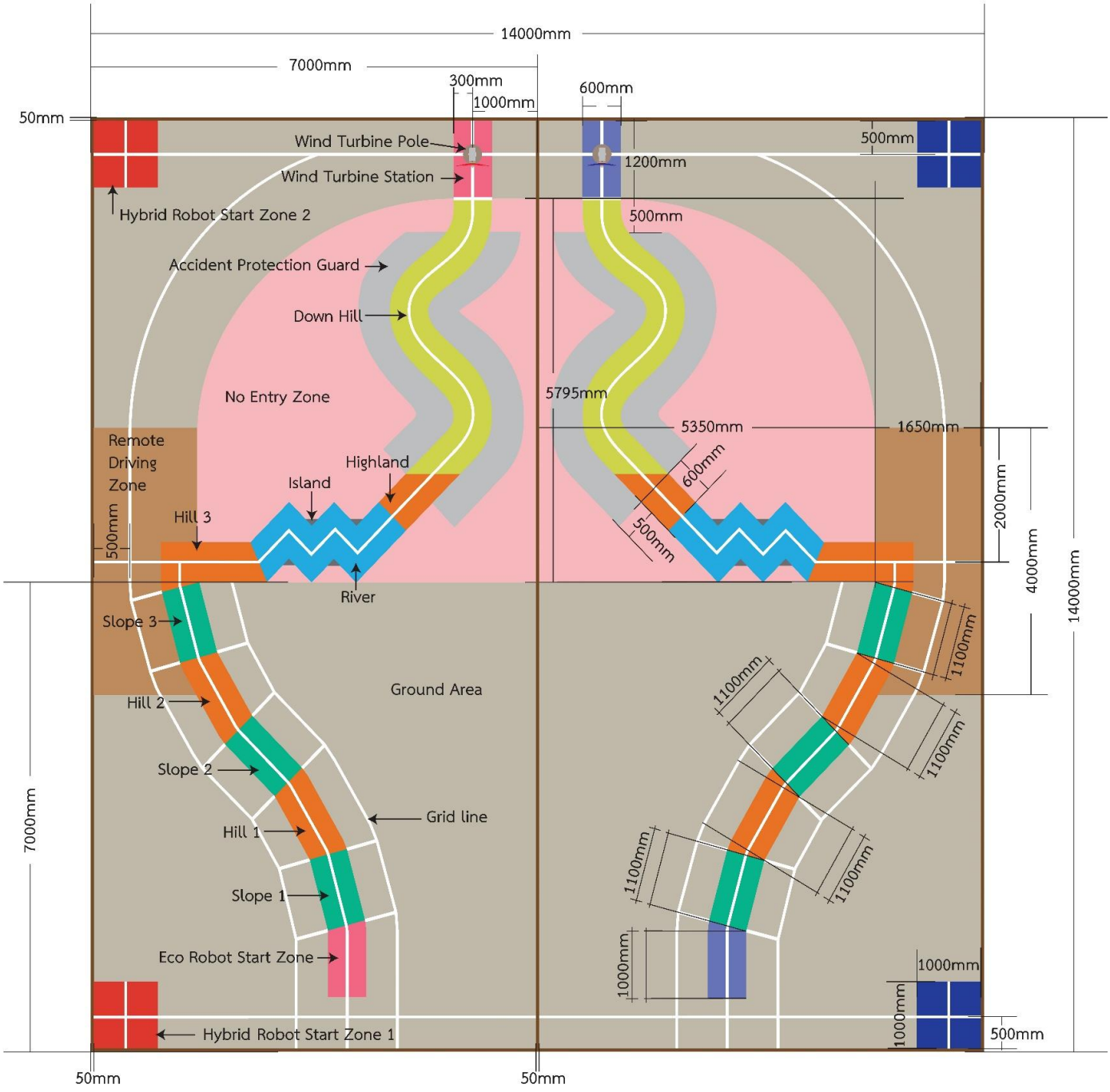


Figure 1: Game Field

OPENCV SETUP IN ANDROID STUDIO

1. Import the OpenCV module from the directory *OpenCV-android-sdk/sdk/java/*
2. In the Project Structure, set Compiled SDK Version to API 23 and Build Tools Version to 23.0.1 for both the app and the OpenCV Library.
3. In the Dependencies of the app, add the OpenCV Library as a module dependency.
4. Copy the native libraries to the project, from *OpenCV-android-sdk/sdk/native/libs/armeabi-v7a* to *<Project Directory>/app/src/main/jniLibs/armeabi-v7a.*
5. To the Manifest, add the *uses-permission android.permission.CAMERA*, and add the *uses-feature android.hardware.camera*.
6. Statically initialize OpenCV by adding the following line of code in the Main Activity class: *static {System.loadLibrary ("opencv_java3"); }*.
7. Add JavaCameraView to the *layout. JavaCameraView extends CameraBridgeViewBase.*
8. Import OpenCV Libraries including Android libraries for CameraView and loading, Core libraries for basic operations and data structures, and Image Processing library.
9. Implement the Main Activity class as CameraViewListener2. Implement its methods - onCameraViewStarted, onCameraViewStopped, onCameraFrame.
10. Instantiate an object of CameraBridgeViewBase, that extends SurfaceView.
11. Link the CameraBridgeViewBase object to the JavaCameraView in the layout. Set maximum frame size for the object.
12. In the onCameraViewStarted method, allocate Mat objects for processing images. In the onCameraViewStopped method, rellease the allocated Mat objects.
13. Process the input frame received by onCameraFrame method everytime an image is captured. Return a Mat object of the image to be displayed on the screen.

USB HOST COMMUNICATION SETUP IN ANDROID STUDIO

1. To the Manifest, add the uses-feature android.hardware.usb.host
2. Create an xml file device_filter.xml and add vendor-id of the accessory to deliver intent only for that specific hardware.
3. To the Manifest, add intent-filter and meta-data for the action USB_DEVICE_ATTACHED with the xml device_filter as the resource. Add intent-filter for the action USB_DEVICE_DETACHED.
4. Create a method connectUsb to be executed at application startup, when device is attached, or when the user grants permission. It comprises the following functionality.
5. The method searchEndPoint - gets the device list of attched devices and stores the desired device in UsbDevice object, gets the usb interface of the device, and gets the bulk transfer input and output endpoints from the interface.
6. The method setupUsbComm - if permission has been granted, usb device connection is opened on the device, interface is claimed by the connection, and setup bytes are sent to the accessory through control transfer. If permission has not been granted, a broadcast is launched for the intent ACTION_USB_PERMISSION.
7. Create a method releaseUsb to be executed when the device is detached. In this method, the interface is released from the connection and the connection is closed.
8. An array of bytes are sent to the usb accessory via usb bulk transfer.

ASYNC TASK Class

Since complex operations and processing cannot be done on the UI thread. The AsyncTask class is used. When an object of a class extension of AsyncTask is instantiated, a new thread is spawned. On this thread, the complex operations are carried out. The UI thread is updated from this thread by putting toasts on the message queue of the UI thread.

LINE SEGMENTATION & FOLLOWING ALGORITHM

1. The captured input image is thresholded.
2. From the resulting binary image, contours are detected.
3. The contour of largest size is extracted from the set of contours. This contour is assumed to be the line that is to be followed.
4. The extrema are used to calculate the angle of line with respect to the current position of bot, and the offset of line.
5. These are the angle and offset errors. A weighted error sum is calculated using the two errors.
6. PID control algorithm is applied on the error to generate the correction angle.
7. The angle byte is sent via USB bulk transfer to the Arduino Microcontroller.
8. The Arduino commands the servo motor to rotate at the calculated correction angle.
9. A toast of the angle is put on the message queue of UI thread.

REGION SPECIFIC ALGORITHMS

PID parameters and error weights are varied according to the region of arena to adjust to the different configurations like straight, zig-zag, and sinusoidal.

I. UPHILL REGION

This is the part of arena before the 90 degree turn at the Hill 3. The grayscale image is thresholded and contours detected as in the general algorithm. This is executed up until Hill 3 is detected by detecting the T. While following the line, any line in the view of camera has 2 extrema. But when T appears in the view of camera, 3 extrema of T are detected. This is used to trigger the algorithm which is to be executed post Hill 3.

II. T REGION (90 DEGREE TURN)

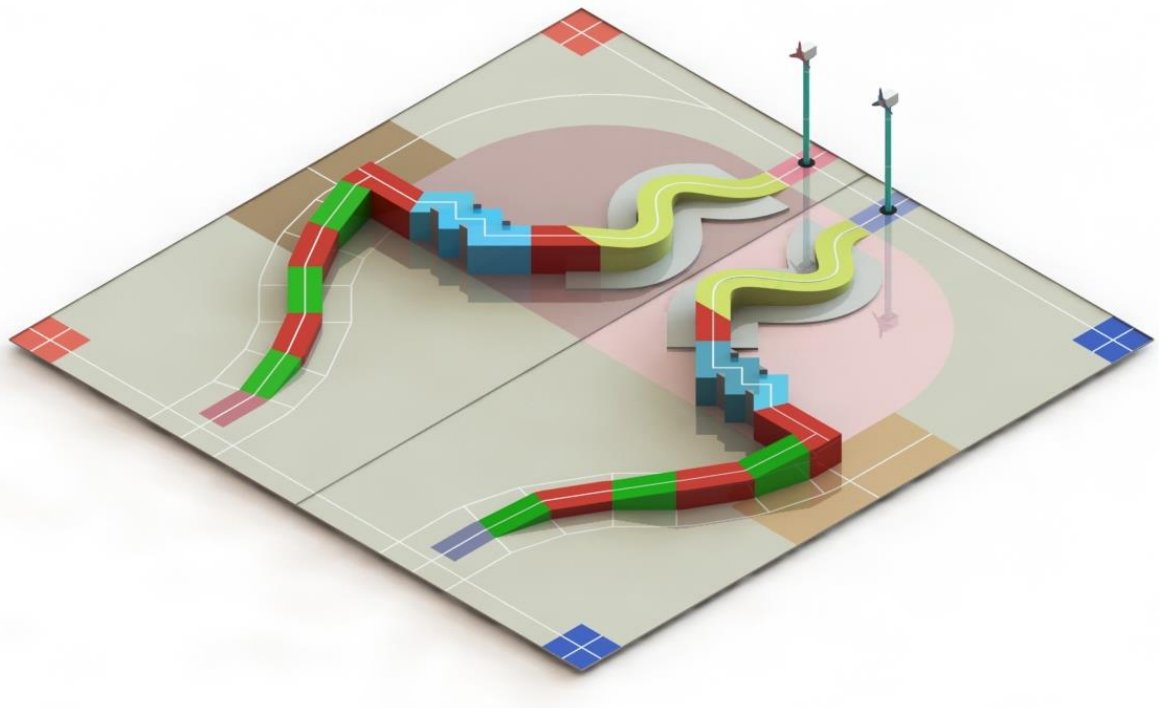
The lower straight part of T is followed until the T descends by a predefined threshold height in the frame. During this course, the speed of bot is estimated by taking the difference of offsets of the top of T between frames. When the top offset exceeds the predefined threshold, a turn angle is calculated based on the estimated speed. The servo is turned at this fixed angle until the River Region is detected. While turning on T, more than 2 extrema are extracted by the segmentation algorithm. Once the bot gets back on straight line, segmentation algorithm detects 2 extrema on the right part of the frame. This is how River is detected and the corresponding algorithm triggered.

III. RIVER REGION (ZIG-ZAG)

In this region, the blue part is removed by thresholding in the red channel. Then the external pink part (the ground surface) is removed by thresholding in the green channel. The result of the two is conjoined bitwise. PID is applied as in the general algorithm. Here, it optimizes the zig-zag path into an almost straight path as we take only the extrema to calculate angle and offset errors. Since the background is pink with high intensities in all 3 channels, Background White Removal algorithm is used to keep track of the line. The transition to Downhill Region is detected through HSV change.

IV. DOWNHILL REGION (SINUSOIDAL SLOPE)

Thresholding is done in blue channel as it is most appropriate for the color of Downhill. The left and right extreme servo angles are predefined, which are larger than those achieved by PID control. This is done to make the bot take a sharp turn on the large turns, where the white line goes out of the camera's view. To avoid interference by background white (the ground surface), 80 columns are sliced from either side of the frame. This reduction in frame width causes the angle and offset measurements to fluctuate. To counter this, Moving Average smoothing filter is applied on the error and then PID is applied. Since the background is white, Background White Removal algorithm is used to keep track of the line. The transition to Station Region is detected through HSV change. In Station Region, the bot is made to continue on the servo angle it was moving on before entering Station, and the angle isn't varied until it hits the pole at the end.



Isometric View

BACKGROUND NOISE REMOVAL (TRACKING ALGORITHM)

This algorithm is used to keep track of the white line in the presence of significant background noise (due to ground surface being white in color). This is a tracking algorithm which computes the expectation of the position of line based on its current position.

1. The 480x320 frame is divided into 3 parts - columns 0 to 199 (left part), 200 to 279(neither left nor right), and 280 to 479 (right part). If any part of the line is in 200-279, the line is considered to be in center.
2. If line is validly in center, the expectation is null.
3. When the line then moves to the right part, the expectation is converted to Right.
4. With this expectation of Right, if the line moves to the left part without passing through center, the left is ignored, the previous angle is retained, and the expectation still remains Right.
5. Now if the line moves into center from left having expected value of Right, the center is ignored retaining the previous angle and expectation still remains Right.
6. The previous angle is retained until the bot returns to follow the line and the line is back in the right part. Then the normal line following resumes.
7. When line moves from the right to center, expectation becomes null and normal line following continues.
8. When line moves from center to left, expectation becomes Left and normal line following continues.
9. In this way, it is able to filter out the white background and keep track of the white line.