A Java-powered FIRST Robot

Noel Poore
FRC Team 1519
Mechanical Mayhem
noel.poore@oracle.com

The Team

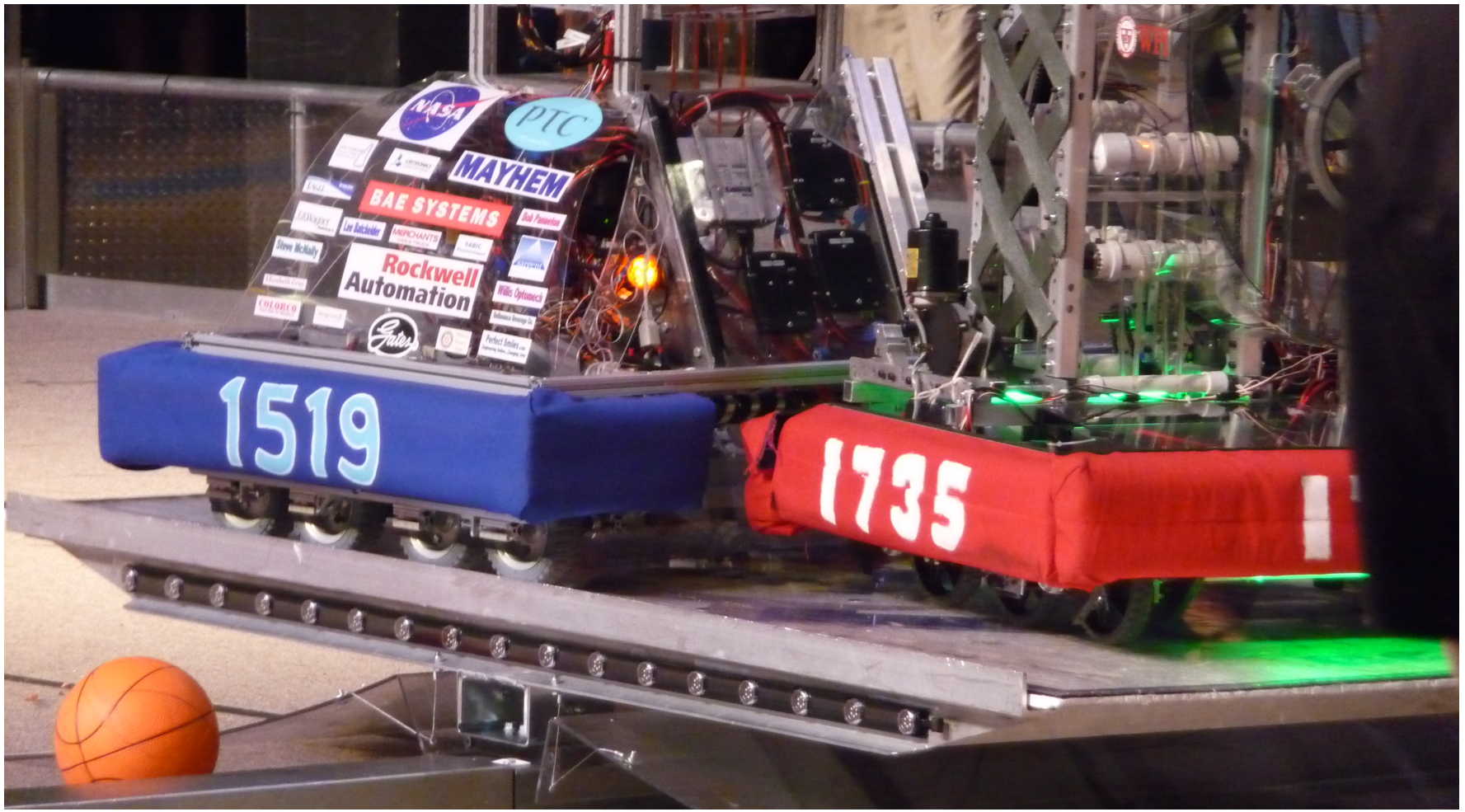

FRC
FIRST Robotics Competition
WINNER
2012
NORTH CAROLINA
REGIONAL
MAYHEM
BAE SYSTEMS
Rockwell
Automation
1519

The Game

- Basketball
  - 8” foam balls
  - Four hoops at each end
- A robot must weigh less than 120lb
- Size less than 38” x 28” x 60”
- Each match is 3 robots against 3
  - 15s autonomous
  - 120s teleoperated

The Game

Balancing



Mechanical Mayhem 2012

- Wide robot
  - takes up less space on bridge, easier to balance
- 8 pneumatic wheels
  - Traverse both bridge and barrier
  - Can hang over the end of the bridge
- Harvest balls from both sides
- Camera-controlled shooter with azimuth, elevation and wheel speed control
- Simple bridge tipper
- Lots of sensors

Robot Software

- Written in Java
- Using WPILibJ on the robot
  - Open source library sponsored by WPI
  - Running IMP (Information Module Profile)
    - “Headless MIDP”
- Using Java SE on the driver station
  - Running on a small laptop under Windows 7
  - We only ever used about 20% of the CPU
- Field provides WiFi wireless network

Subsystems

- Drivebase
- Ball Harvester
- Shooter
- Bridge tipper
- Autonomous
- Camera
- Define your own class – must extend library Subsystem class

Buttons

- The control inputs
- Usually real buttons on a joystick or gamepad
- Can be virtual buttons
  - If tilt sensor says we are tilted by > 15 degrees
  - If harvester motor current > 20A for >1s
- Very easy to change or remap on the fly
  - Makes the drive team happy
- Buttons cause Commands to be scheduled

Commands

```
public class DisabledOnlyJoystickButton extends Button {

    private GenericHID joystick;
    private int buttonNumber;
    private DriverStation ds;

    public DisabledOnlyJoystickButton(GenericHID joystick,
                                      int buttonNumber) {
        this.joystick = joystick;
        this.buttonNumber = buttonNumber;
        ds = DriverStation.getInstance();
    }

    public boolean get() {
        return joystick.getRawButton(buttonNumber) &&
                  ds.isDisabled();
    }
}
```

Commands

- Instructions to a subsystem to do something
- CommandGroup allows you to create a sequence of commands
- We created 65 commands
  - Including debug and calibration commands
  - And Autonomous programs
  - Automatic and manual operation
    - Important to have a manual backup system
    - And have the drive team practice using it

Commands

```
public class SetTarget extends CommandBase {
    private int target;

    public SetTarget(int target) {
        requires(shooter);
        this.target = target;
    }

    // Called just before this Command runs the first time
    protected void initialize() {
    }

    // Called repeatedly when this Command is scheduled to run
    protected void execute() {
        shooter.setTarget(target);
    }
```

Commands

```
    // Make this return true when this Command no longer needs to run
    protected boolean isFinished() {
        return true;
    }

    // Called once after isFinished returns true
    protected void end() {
    }

    // Called when another command which requires one or more of the
    // same subsystems is scheduled to run
    protected void interrupted() {
    }
}
```

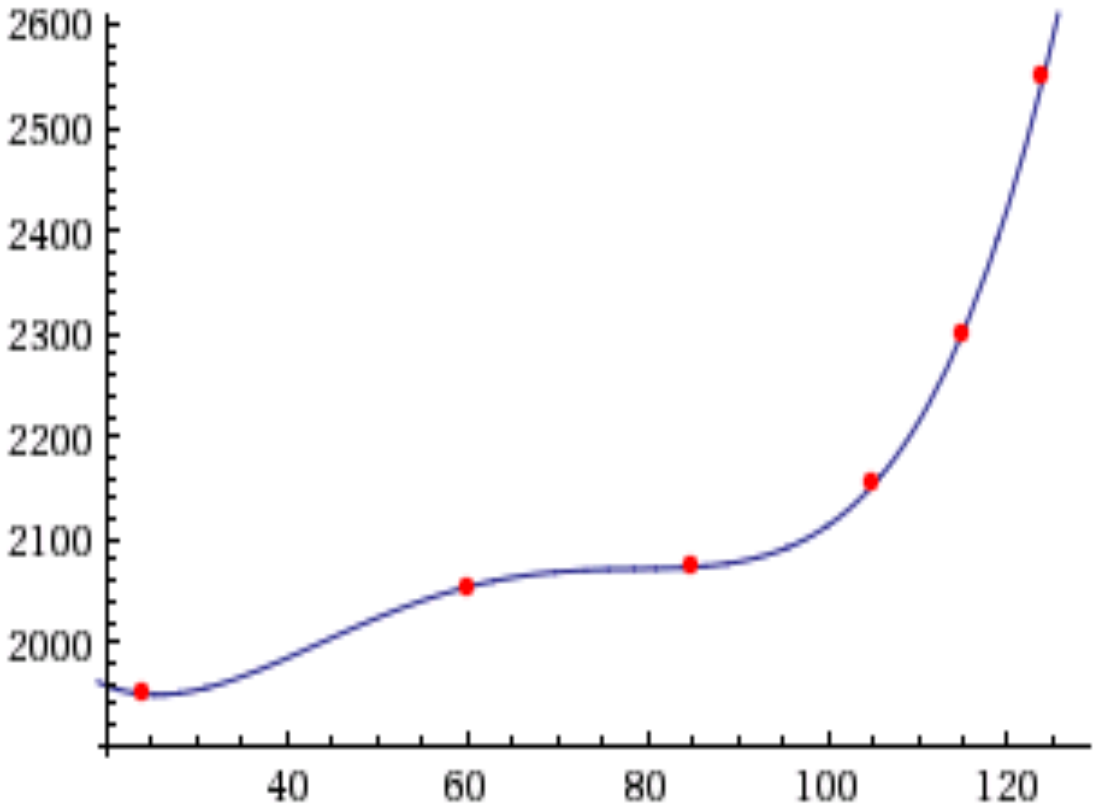
Shooter Targeting

- The hoops have a rectangle of retro-reflective tape above them
- The robot illuminates this with green LEDs
- Webcam video stream sent to the driver station
- Video processed to find vision targets
- Each frame processed independently
- Candidate target coordinates identified
- Target information sent back to the robot

Shooter Targeting

- Robot receives target information (X, Y)
- X value → adjust azimuth PID set point
- Y value → a proxy for “how far”
- Use linear regression to calculate desired shooter elevation and wheel speed
  - Quintic polynomials
  - Thank you Wolfram Alpha!
- Separate PID controls for elevation and wheel

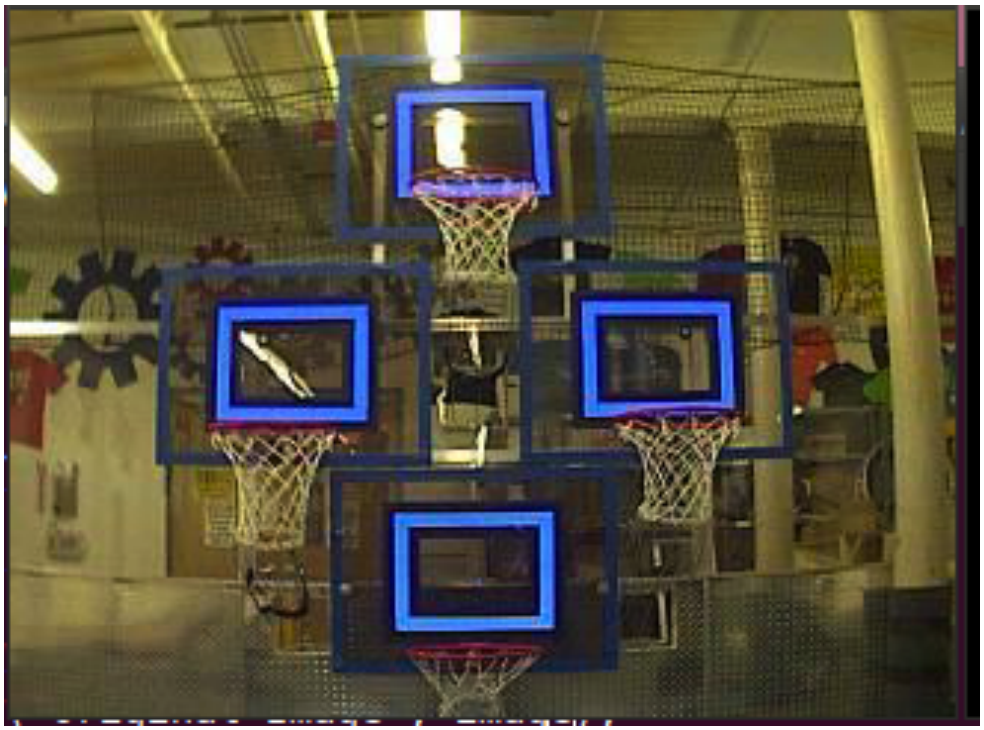
Shooter Wheel Speed



Shooter Azimuth Control

- Shooter position measured using an encoder
- Camera X value converted to encoder offset
- Desired shooter azimuth =
  current azimuth + delta from X value
- Need to account for time lag
- We keep an “azimuth history”
  - So we can look up what the “current” azimuth was when the picture was taken

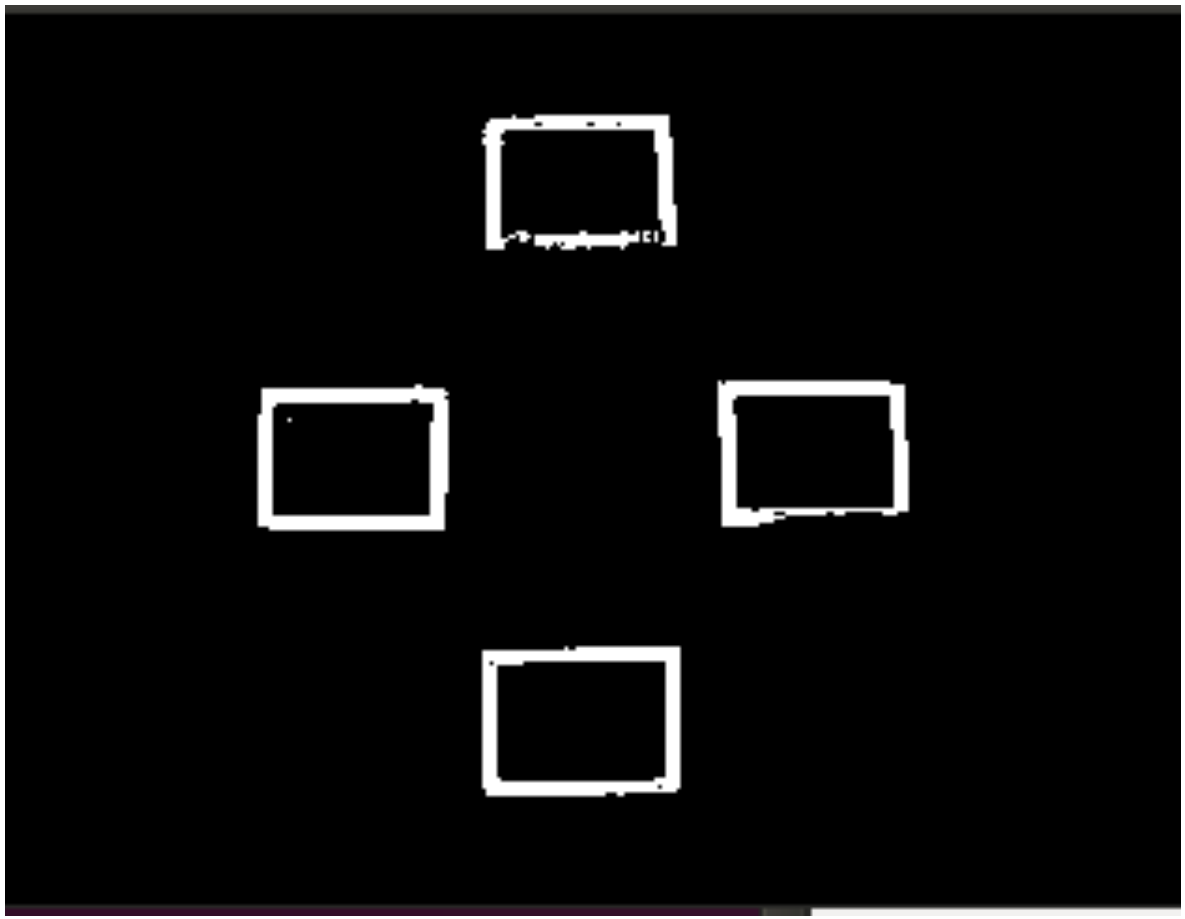
The original image



Processing the Images

- Images are converted to HSV
- Then a color threshold is applied to convert to a binary image
  - Beware of cvInRangeS() upper threshold
- This image has gaps because of the hoops

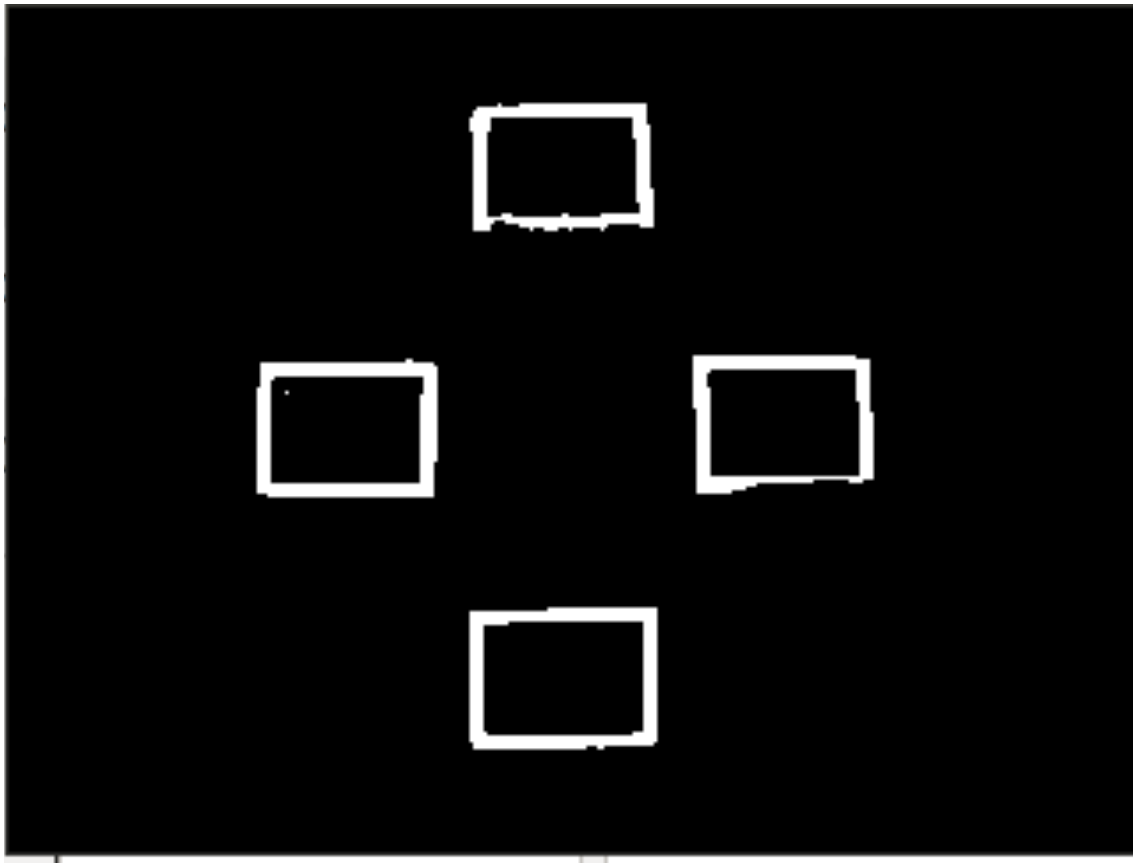
Processing the Images



Processing the Images

- The image is then closed
  - Dilate then erode
  - Grow the white area then selectively shrink it again
- This fills in most of the gaps
  - The rest of the gaps are dealt with later

Processing the Images



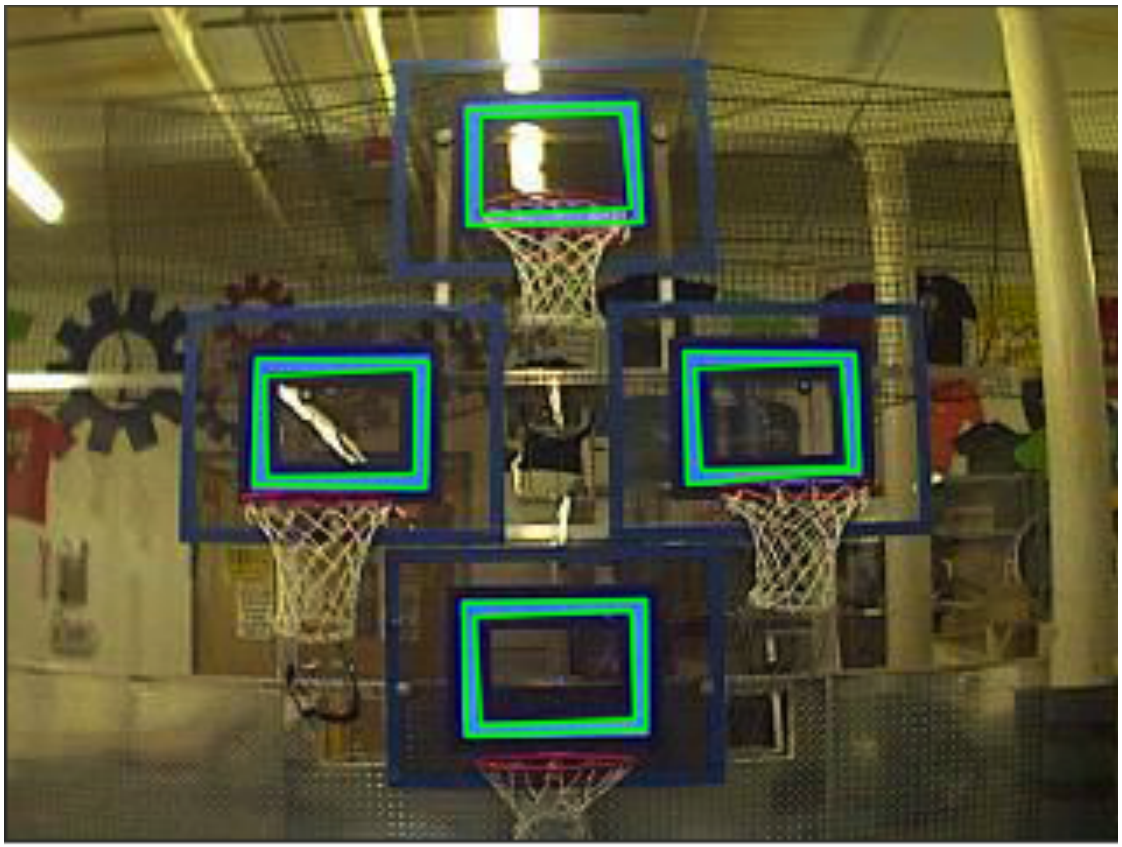
Processing the Images

- Contour detection
- Convert contours to polygons
- Approximate the outline of the polygon
  - Use heuristics to complete broken outlines

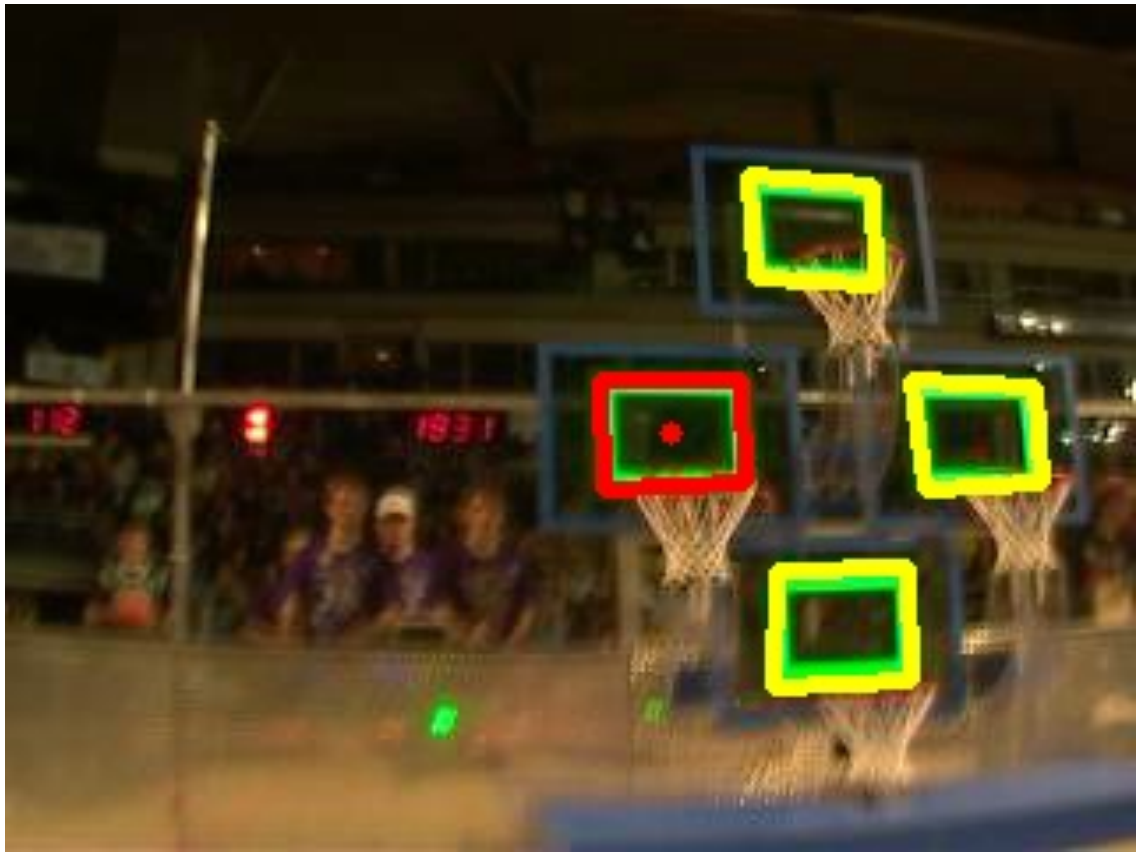
Processing the Images

- Keep polygons which have
  - Right size, # corners, angles and aspect ratio
- Discard “inner” polygons
- Did not have any problems with false positive target recognition

Processing the Images



What the Drivers See



Shooter automatic mode

- Shooter will track the target whenever visible
- Operator can manually adjust azimuth
- Soft and hard limit switches
- Operator presses “Fire when ready” button
- Robot will shoot when
  - Shooter on target
  - Target Y value within limits
    - Depends on which hoop
  - Robot more or less stationary

Match Video

Sensor Usage

- Encoders
  - Drive wheels – drive straight and measure distance
  - Shooter wheels – PID control of wheel speed
  - Shooter hood – PID control of elevation
  - Azimuth – PID control for camera targeting
- Heading gyro used for autonomous spins
- Tilt
  - Gyro & 3-axis accelerometer
  - Use Kalman filter to reduce noise and eliminate gyro drift
- Analog potentiometer
  - Bridge tipper position
- Infra-red distance sensors to detect balls and stop harvester

Kalman Filter

- Used to mix tilt gyro and accelerometer readings to get a more reliable tilt reading for the robot
- Removes high frequency noise from the accelerometer readings
- Removes low frequency noise (drift) from the gyro readings
- Results in a much more reliable tilt reading
  - Important for automatic balancing

Automatic Balancing

- Driver presses joystick button to initiate and controls speed of robot up the bridge
- Robot code monitors tilt to automatically position the robot at the right balance point
- Code will automatically move the robot back (and forwards as necessary) to find the balance point
- Camera lights flash when bridge is balanced
- Can rebalance if another robot moves

Bridge Tipper

- Simple mechanism
- Window motor moves arm, servo moves hook to relieve stress on motor
- Software control uses 6 stage state machine
  - To synchronize tipper position and servo-controlled hold-down
- Operator presses one button to deploy, a second button to retract
- Driver can also toggle bridge tipper when necessary

Call for Action

- If you have children, get them involved in FIRST Lego League or FIRST Robotics Competition
- Get involved as a coach
- Help find sponsors
- Have fun teaching high school students how to use Java

Commands

```
public class TipOurBridge extends CommandGroup {
    public TipOurBridge(int firstGoal, boolean gear) {
        addSequential(new AzOffset(-348));
        addSequential(new ChangeGear(Drive.HIGH_GEAR));
        addSequential(new SetTarget(firstGoal));
        addSequential(new WaitForShooterWheels(3.0));
        addSequential(new ShootBalls(2, 3.0));
        addSequential(new SpecialShot(Shooter.SPECIAL_CENTER_AZ));
        addSequential(new HarvesterOnOrOff(Harvester.ON));
        addSequential(new DriveStraight(Drive.HIGH_GEAR, -1.0, 7*12 + 9));
        if (gear != Drive.HIGH_GEAR) {
            addSequential(new ChangeGear(gear));
        }
        addSequential(new SpinBy(55, gear));
        addSequential(new BridgeTipperInOrOut(BridgeTipper.POS_DEPLOYED));
        addSequential(new WaitForBridgeTipper(BridgeTipper.DEPLOYED));
        addSequential(new DriveStraight(gear, -0.5, 4.0*12));
```