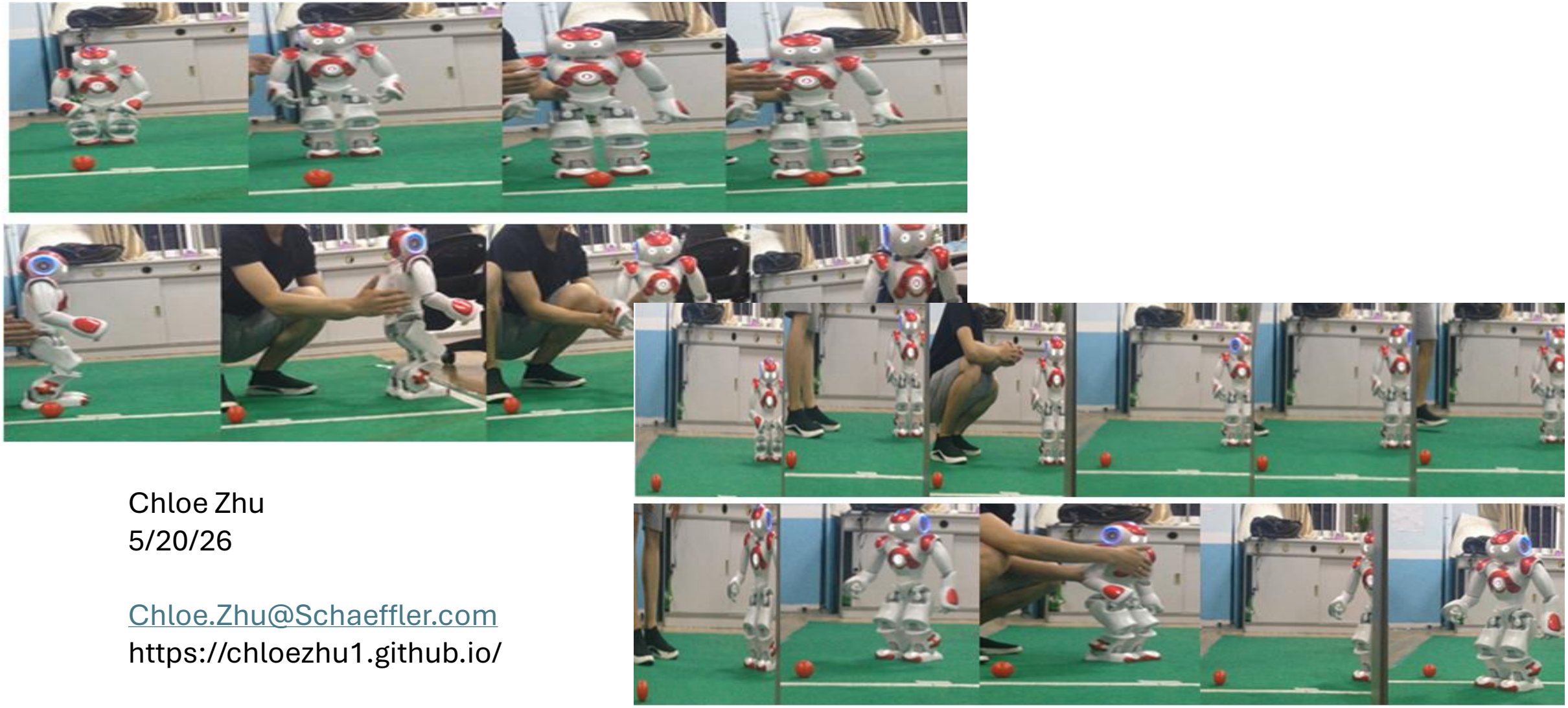
Computer Vision, Motion Control and Communication Design of Autonomous Robots based on Embedded Linux



Chloe Zhu
5/20/26

Chloe.Zhu@Schaeffler.com
https://chloezhu1.github.io/

Agenda

- Computer Vision
- Motion Control
- Communication (Precise Synchronous Frequency Offset Apparatus)
- Embedded Linux Application in Robotics

Embedded Linux in NAO Humanoid

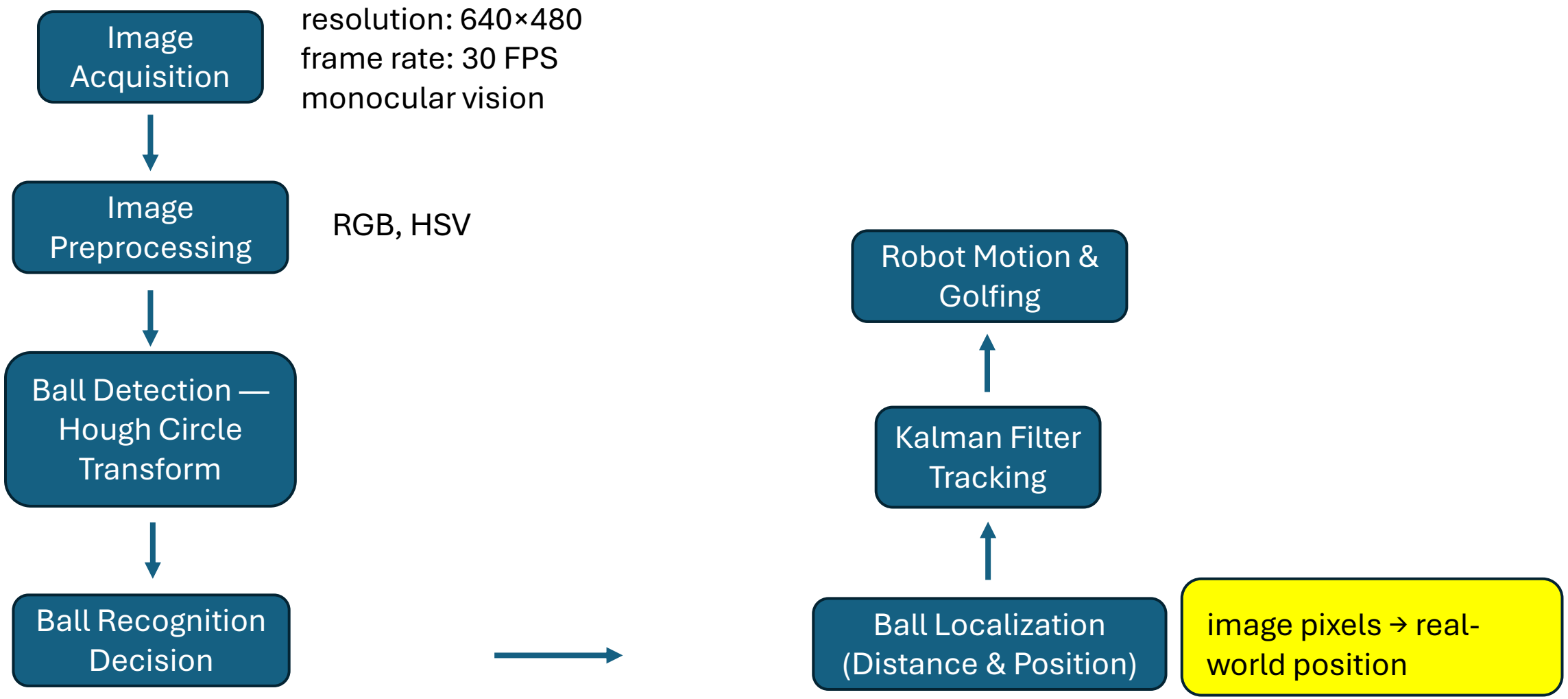
- The NAO robotics platform, originally developed by Aldebaran and SoftBank, and now by Maxtronics.
- OpenNAO operating system is based on the Gentoo embedded Linux OS, and uses the NAOqi API for autonomous robot control.
- We also used the OpenCV computer vision library as part of our open source software stack to program our NAO humanoid robot.

Robot Golfing Competition

YouTube Link of our robot

Nao humanoid robot is an autonomous, programmable humanoid robot.

Complete Technical Pipeline



Full Coordinate Transformation Pipeline
Ball Detection → World Position

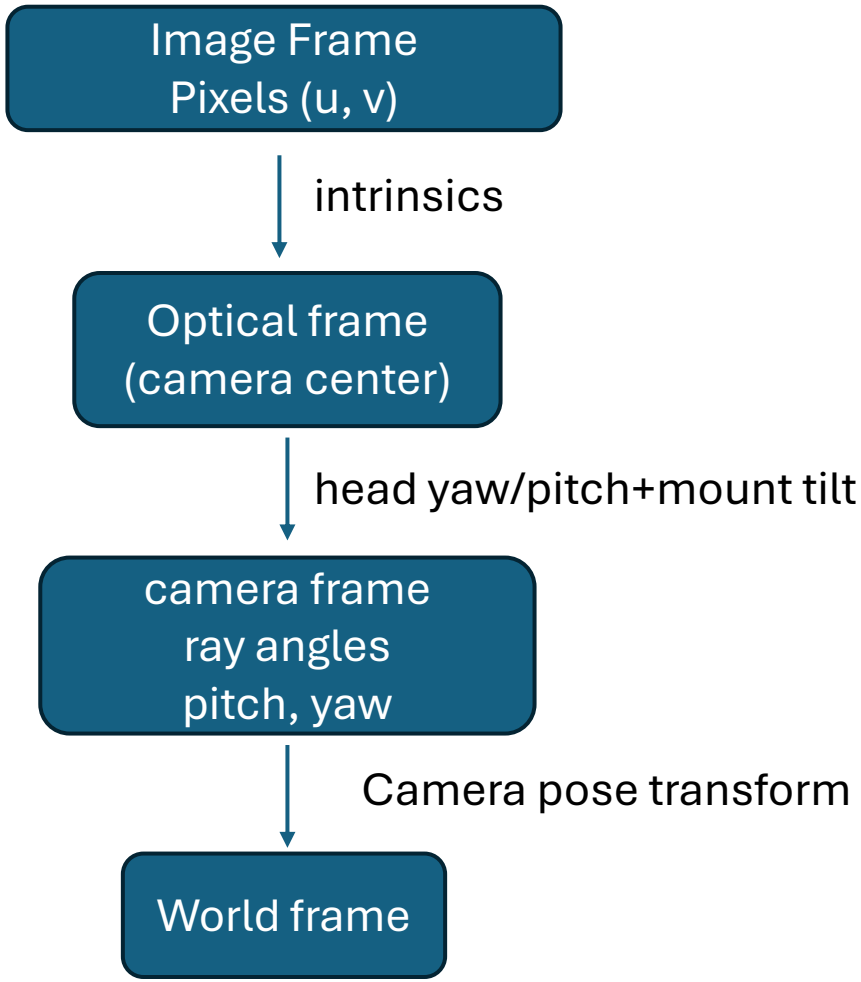


Image Coordinate System (Pixels)

(center_x, center_y, radius)

image center = (W/2, H/2)

Pixel Offset → Camera Angles

img_pitch = (center_y − img_center_y)/H * cameraPitchRange
img_yaw = (img_center_x − center_x)/W * cameraYawRange

Add Head + Camera Mount Angles

camera_pitch = head_pitch + camera_mount_tilt
ball_pitch = camera_pitch + img_pitch
ball_yaw = head_yaw + img_yaw

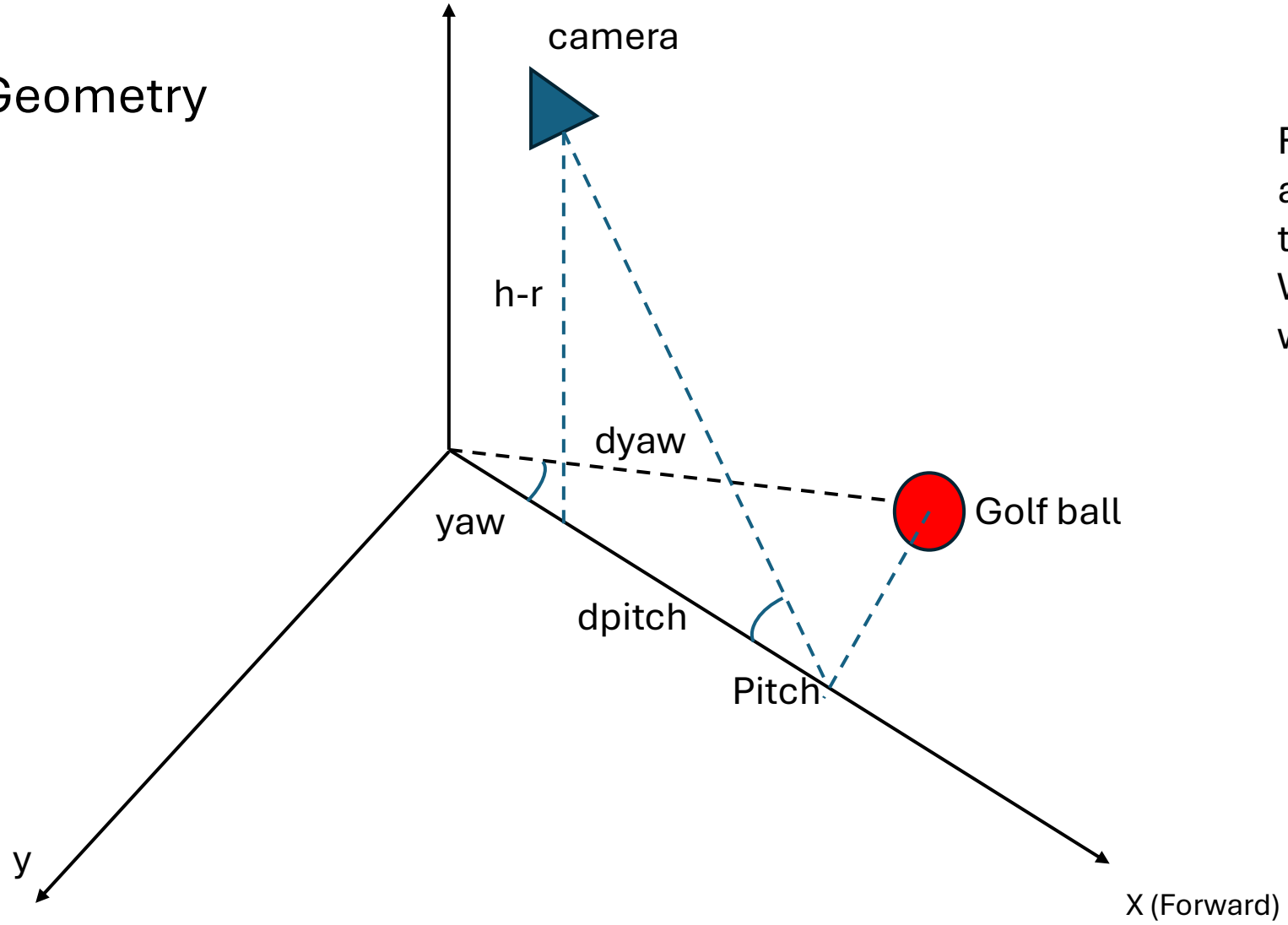
Polar → Robot XY Coordinates

disX = distance * cos(ball_yaw)
disY = distance * sin(ball_yaw)

Camera Frame → World Frame

ball_world_X = cameraX + disX
ball_world_Y = cameraY + disY

Geometry



For quick illustration, I set the x-axis along the pitch distance direction in this example. In general, it's not true. We need to consider X_camera as well.

Converting camera frame to world frame.

Compensating y-coordinate in world frame

Physics model
angles → tan → cos → geometry

+

★ **Empirical model**
ky polynomial correction

```
ky = a*ballX**4 - b*ballX**3 + c*ballX**2 - d*ballX + e
ballY = ky * ballY
```

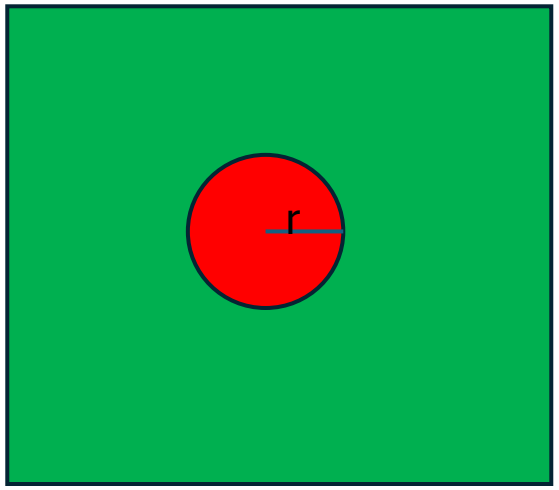
4th-degree polynomial correction factor applied to Y (lateral distance)

data-driven error compensation

In real robots, geometric model has systematic errors:
camera tilt not exact
head angle offsets
lens distortion
FOV scaling not perfectly linear
camera mount flex
joint encoder bias
ground plane not perfectly flat
image center not perfectly calibrated

1. Collect calibration data: trueX, trueY (measured by tape / motion capture). Run vision system →compute: estX, estY
2. Compute Y error ratio: ky_needed = trueY / estY, as a function of distance: ky_needed = f(estX)
3. Fit polynomial: regression

Computer Vision: How to select the red ball from the detection?



Area_ball/area_square = 0.196

rRatio = rScore / total_pixels
gRatio = gScore / total_pixels

Image Processing flow

Selection Criteria:
if rRatio >= 0.12
and gRatio >= 0.1
and abs(rRatio-0.19) < abs(rRatioMin-0.19):
Condition A — enough red pixels
Condition B — enough green pixels
Condition C — red ratio close to target value

Grab the latest camera frame → Set circle detection constraints → Build a “ball-likelihood” grayscale/binary image

Detect circles in that processed image → Select the best circle candidate → Handle “no ball found”

Select the best circle candidate → store ball data and compute position

Computer Vision:

Red ball detection:

- Crop the image
- Convert BGR → HSV: frameHSV = cv2.cvtColor(frameArray, cv2.COLOR_BGR2HSV)
- HSV threshold (binarization): frameBin = cv2.inRange(frameHSV, minHSV, maxHSV)
- Erosion: frameBin = cv2.erode(frameBin, kernelErosion, iterations=1)
- Dilation: frameBin = cv2.dilate(frameBin, kernelDilation, iterations=1)
- Gaussian blur: frameBin = cv2.GaussianBlur(frameBin, (9,9), 0)
- Return processed mask

remove small noise while preserving shape

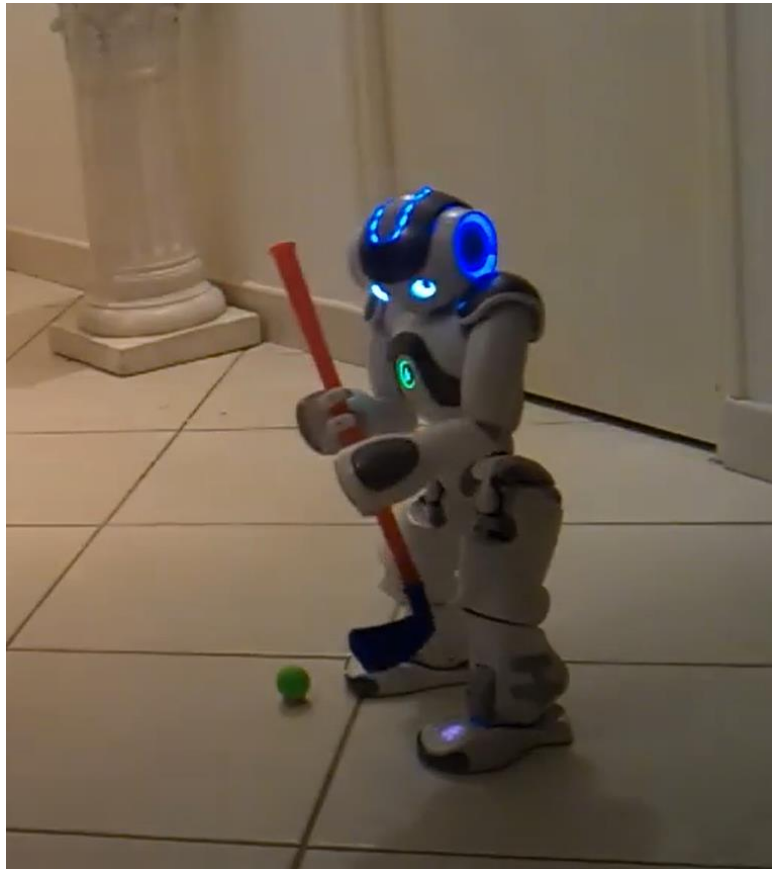
Yellow Stick detection:

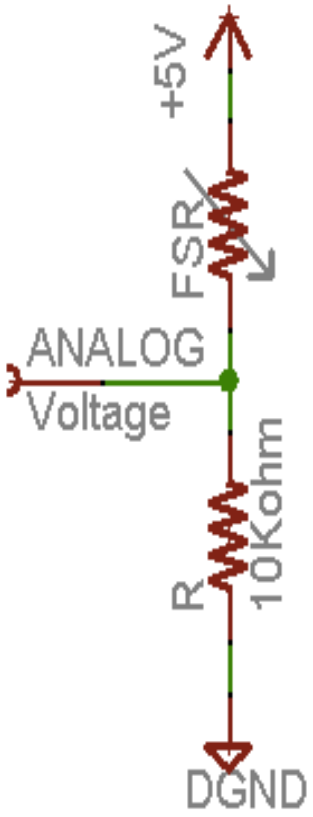
- Capture and crop the image
- Find contours in the binary image: _, contours, _ = cv2.findContours(frameBin, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
- Loop through contours and filter by size: Compute perimeter and area, and filter by Size (Only keep contours large enough to be real objects). Get their bounding rectangles.
- Shape filter — stick-like aspect ratio: rects = [rect for rect in rects if (1.0*rect[3]/rect[2]) > 0.8]. Recall: rect = [x, y, w, h]
- Convert to numpy array

Continued:

- Choose the “most stick-like” rectangle: Selects the rectangle with **largest height/width ratio** → most vertical
- Adjust Y coordinate back to full-frame space

Closed-loop feedback control using Force-sensitive Resistor (FSR)





Principle: find the robotic I/O layout of analog, GND and 5V (as power). Connect FSR from 5V output, in serial to 10K resistor, then ground. Having analog (ADC) trace the voltage.

Via the circuit and Nao's Arduino data logger, I could record FSR voltages at regular intervals, thus having pressure information at self-defined sampling rate, to control and adjust the coordinates of the robotic arm hitting the golf ball.

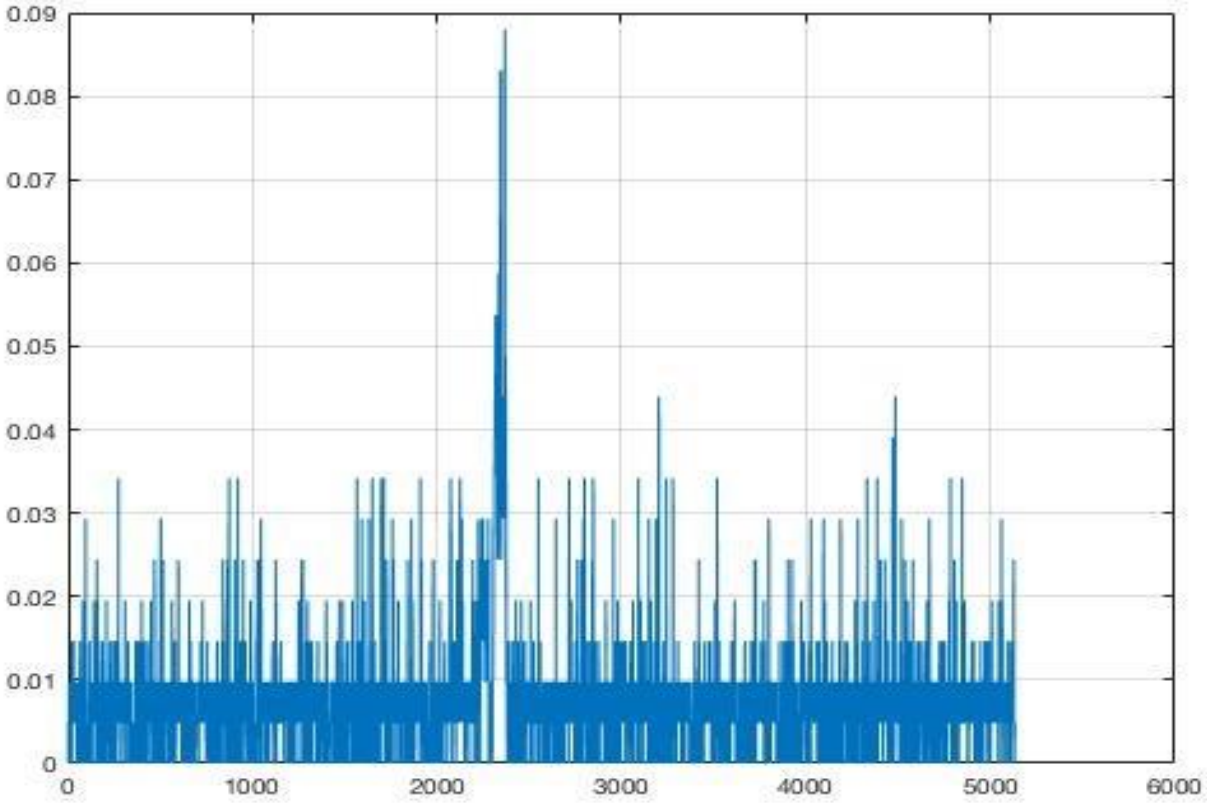
Correlation between the coordinates and the Force (Voltage/Resistance)

- Voltage – pressure
- We transfer pressure to a corresponding visible parameter – coordinate. We have x,y,z 3D coordinate in this application. We can break down the coordinates each direction. For simplicity, we take x-direction for demonstration, so we can control x coordinate to control the pressure applied when playing golf.

Measurement and Data Logging of the Output Voltage

The analog ADC input to Arduino Uno board of Nao for voltage reading. I used the same Arduino Uno board as in the robotic system, and wrote the Matlab script to plot and analyze the relative ADC counts (y-axis) wrt time (x-axis/ms):

```
a=arduino('/dev/cu.wchusbserial1420','uno');
interval=10000;
passo=0.05;
x=[];
t=1;
while (t<interval)
    b=readVoltage(a,'A0');
    x=[x,b];
    plot(x);
    grid
    t=t+passo;
    drawnow;
end
```



By setting passo value, I could manage the interval of the sampling.

Communication Design

What is an effective way to use RF communication without interference with other transmitters? How do we get precise synchronous frequency signal?

In robotics competition/robotics stage performance, if we use RF communication module, there might be more than 10 robotic teams communicating with robots at the same time, and each team might have multiple robots to control (think of a football team).

Precise Synchronous Frequency Offset Apparatus in Robotics Application

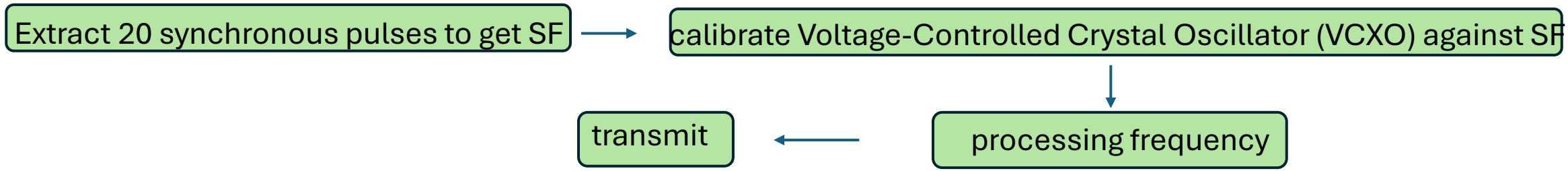
The Precise Synchronous Frequency Offset Apparatus is an intelligent RF calibration system for robotic communication modules. Using a microcontroller-based architecture and real-time frequency measurement/control, it replaces traditional analog Phase-Locked Loop (PLL) synchronization methods to achieve high-precision synchronous frequency calibration.

Solution:

Frequency calibration: Standard Frequency (SF) signal 1(MHz) from China Central Television (CCTV).

Interlaced scan
Phase-Alternating Line (PAL) – based TV (First scan odd-numbered lines, then even-numbered lines, in total 625H, 312.5H each.)

At 16H and 329H of any CCTV programs, they contain 20 synchronous pulses from cesium atomic clock whose precision can reach 10^{-11} – 10^{-15}.



Thus, we can assign the frequency to each robot from 200 MHz, step size as 1 MHz increment, e.g. the second robot RF communication can be set as 201 MHz, the third 202MHz, etc.

Connection to Embedded Linux

Embedded Linux System

↓

NAOqi Middleware

↓

Python Robot Control + Computer Vision

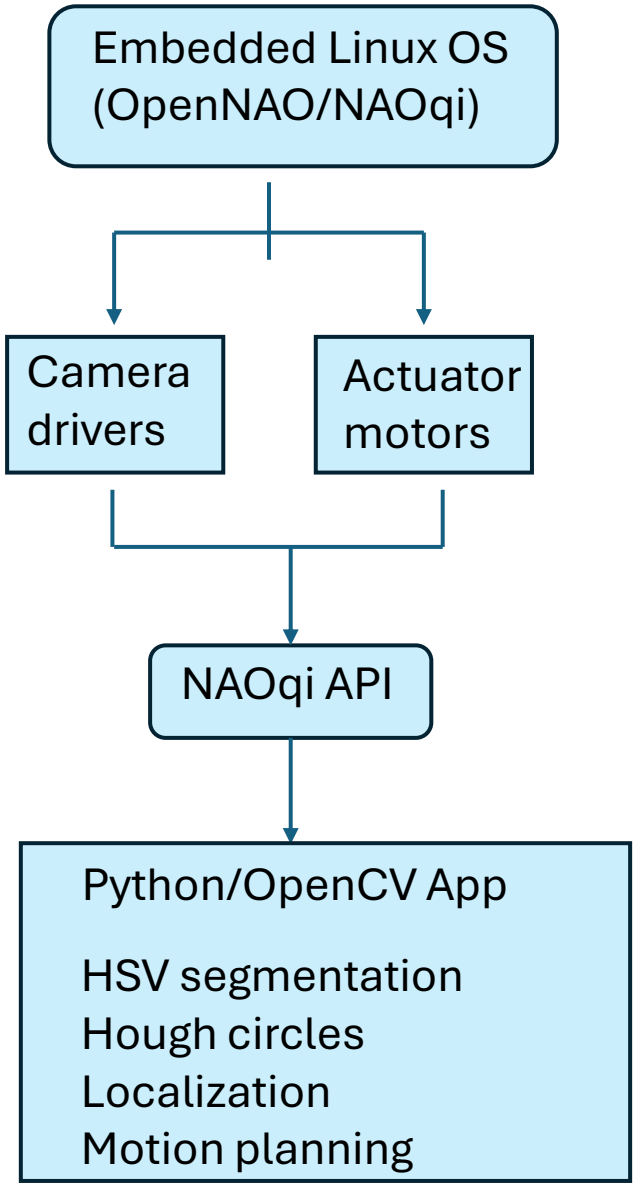
↓

Camera / Motors / Sensors

↓

Autonomous Robot Behavior

Architecture Diagram



- NAO Robot Runs an Embedded Linux OS: OpenNAO / NAOqi OS (based on Gentoo)
- Vision Pipeline Uses Embedded Linux Camera Interfaces:

Camera Sensor
↓
Linux camera driver
↓
NAOqi camera API
↓
Python/OpenCV
↓
Vision algorithms

The Linux-based robot OS handles:
actuator communication
scheduling
sensor polling
safety constraints
synchronization

- Motion Control Uses Embedded Middleware + Hardware Drivers

motionProxy.setAngles(...)
motionProxy.moveTo(...)

These are high-level calls into:

Python API
↓
NAOqi middleware
↓
Embedded motor controllers
↓
Servo actuators

- Real-Time Sensor Fusion on Embedded Hardware
- Resource-Constrained Embedded Computing:

HSV thresholding
Used because:
lightweight
efficient
low compute cost

Hough circles
Optimized circle detection:
avoids heavy deep learning models
suitable for embedded CPU

- Middleware Architecture
- Networking & Inter-Process Communication:

NAOqi modules communicate over:
TCP/IP
proxies
shared middleware APIs

ALMotion
ALVideoDevice
ALMemory

Application Layer
Python vision/motion logic

Middleware Layer
NAOqi APIs

Linux OS Layer
scheduling / networking / drivers

Hardware Layer
cameras / IMU / motors / sensors

Thank you so much!

Chloe.Zhu@Schaeffler.com
https://chloezhu1.github.io/

YouTube Link of our robot