Remote Controlled Segway ev3dev Implementations

June 26, 2020

Lorenzo Agnolucci, Alberto Baldrati, Giovanni Berti

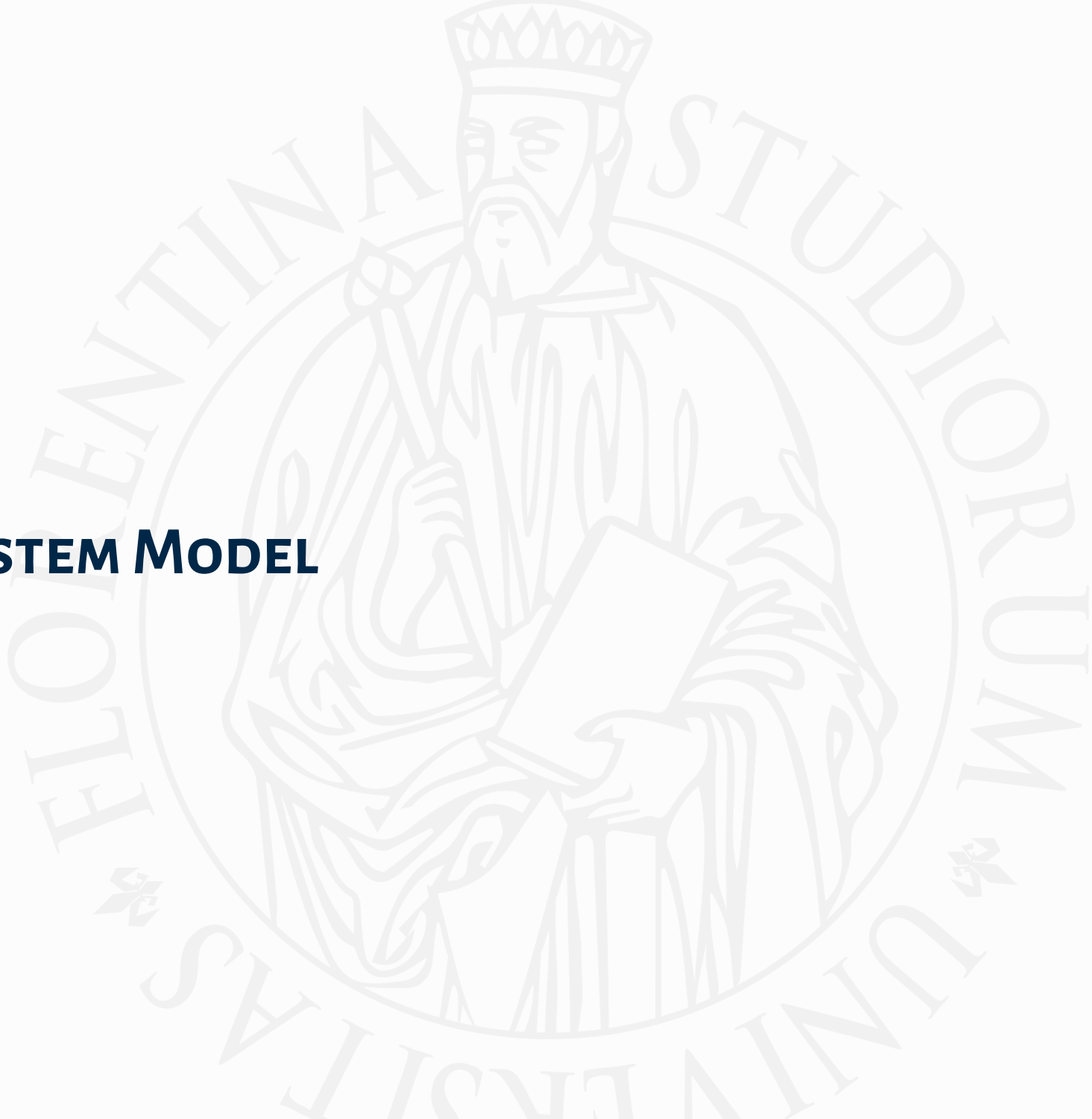
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Firenze

Index

Introduction

Introduction

- A Segway-like two-wheeled robot is a non-linear dynamical system characterized by two equilibrium states:
  - An unstable one which corresponds to the vertical position
  - A stable one which corresponds to the horizontal position of the robot
- The hardware board used to develop the robot is a LEGO® MINDSTORM EV3
- We implemented a control system using the ev3dev operating system
- We extended such model in order to allow free movement in all directions and remote control using an Android app



System Model

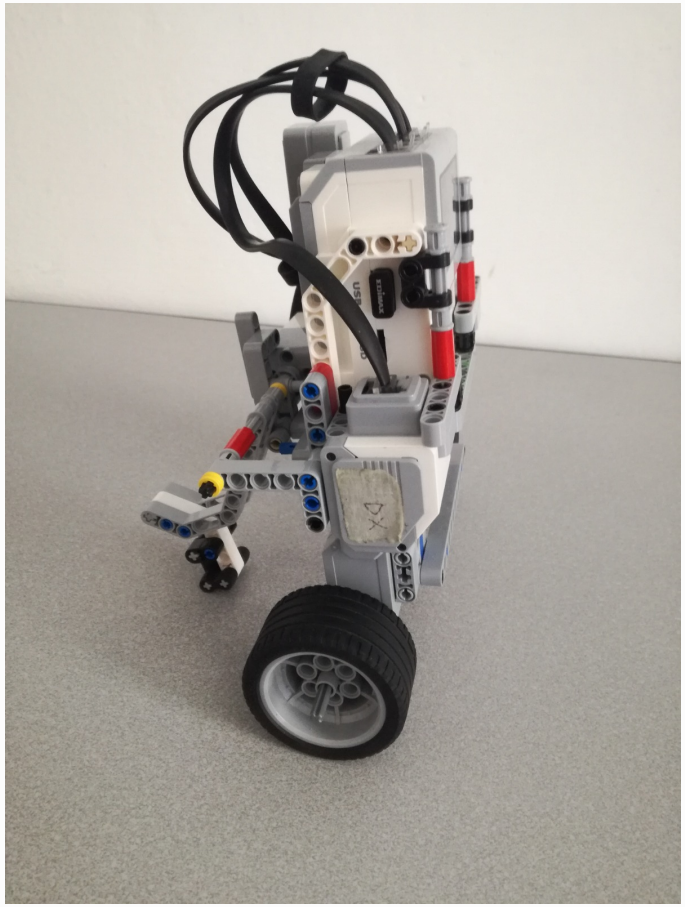
System description

- The system is an inverted pendulum with wheels and with a kickstand
- The system state is described by:

$$x = \begin{bmatrix} \theta & \psi & \dot{\theta} & \dot{\psi} \end{bmatrix}^T$$

  The vertical equilibrium position coincides with $\psi = 0$

- We use the linearized model around this equilibrium state



Control Model

- The control model is derived from the theory of optimal control:

$$u = -Kx$$

where $u = \begin{bmatrix} V_l & V_r \end{bmatrix}^T$ are the voltages given in input to left and right motors and K is the gain matrix

- To make the system more robust we use an augmented state $\tilde{x} = \begin{bmatrix} x(t) & z(t) \end{bmatrix}^T$, where $z(t)$ represents the integral action on θ.

- The gain matrix K is

$$K = \begin{bmatrix} 0.8559 & -44.7896 & 0.9936 & -4.6061 & 0.5000 \\ 0.8559 & -44.7896 & 0.9936 & -4.6061 & 0.5000 \end{bmatrix}$$

Control Model Implementations

General robot operations

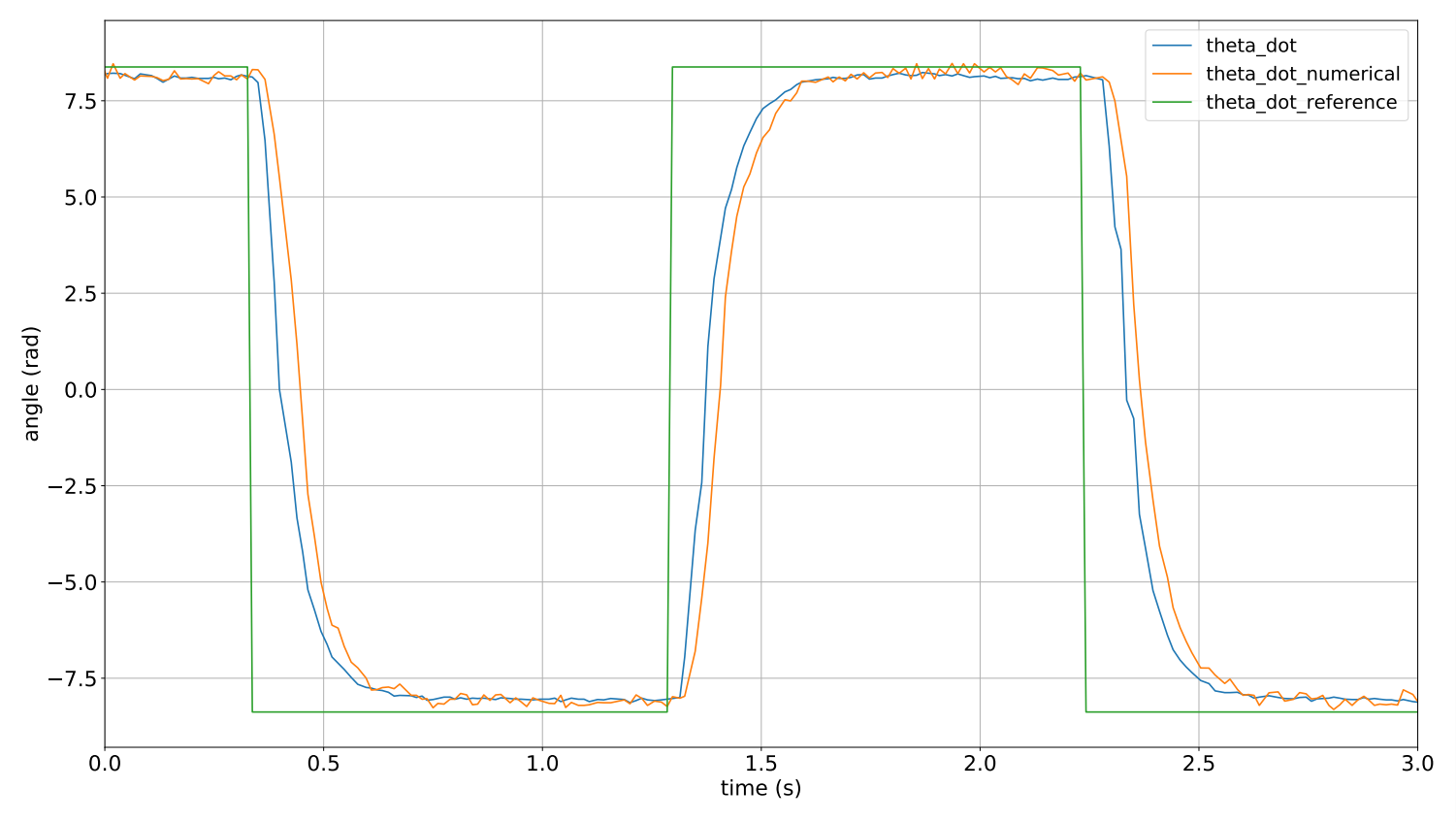
- To implement the control model described we need access to the system's state
- ev3dev allows us to directly probe the values of engine angles (θ) and speed ($\dot{\theta}$), gyroscope angle (ψ) and angular velocity ($\dot{\psi}$) from the board sensors
- To compute integral action on θ we use forward Euler's method:

$$\theta_{int}(n) = \theta_{int}(n-1) + [t(n) - t(n-1)] \cdot \theta(n-1)$$

- When starting the robot it lays upon its kickstand at an angle $\psi = 14°$

General robot operations

- ev3dev provide direct access to the angular velocity of the two engines and the gyroscope angle
- We do not need numerical integration and derivation (unlike MATLAB)



General workflow

1. **Run** the program executable and enter in an **idle** state
2. **Calibrate** the gyroscope for two seconds
3. **Start** the control loop and **lift** the kickstand right after
4. **Keep** the control loop running until requested by the user
5. **Lower** the kickstand, **stop** the control loop and accelerate forward in order to **lay** on the kickstand
6. **Go back** to step 1

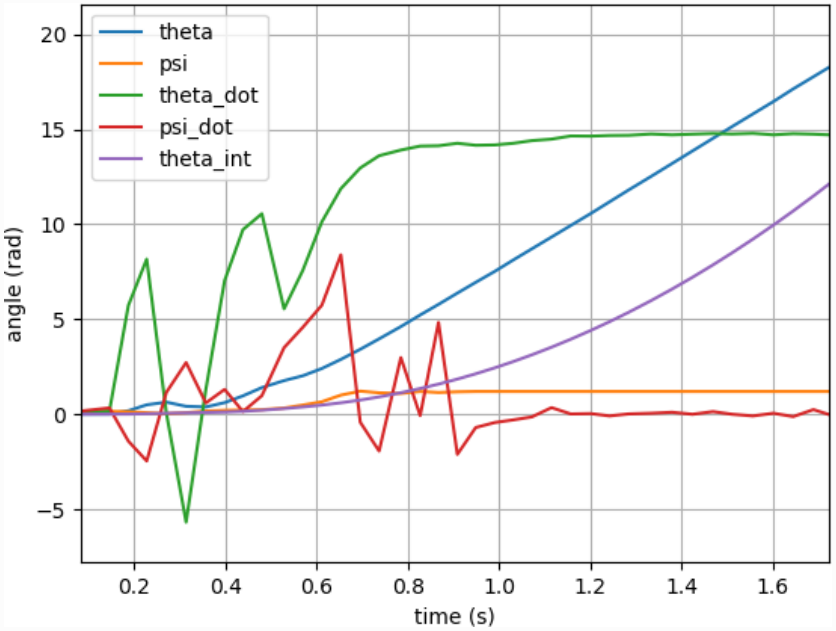
Python implementation
High-level

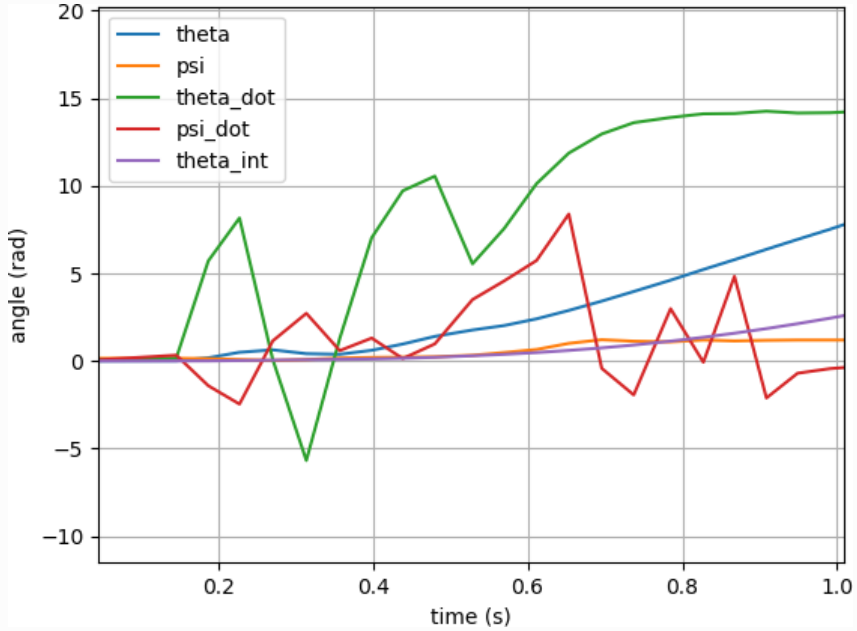
- Based on the `python-transitions` library
- Finite state machine closely resembles the Stateflow abstraction used in MATLAB baseline model
- Sampling times too high, $\simeq$ 250ms
- The control model implementation can not stabilize the robot

Python implementation

Performance oriented

- Manual state management with Python threads
- Manual file handling instead of using ev3dev built-in functions
- Unable to control the robot with $\simeq$ 70ms

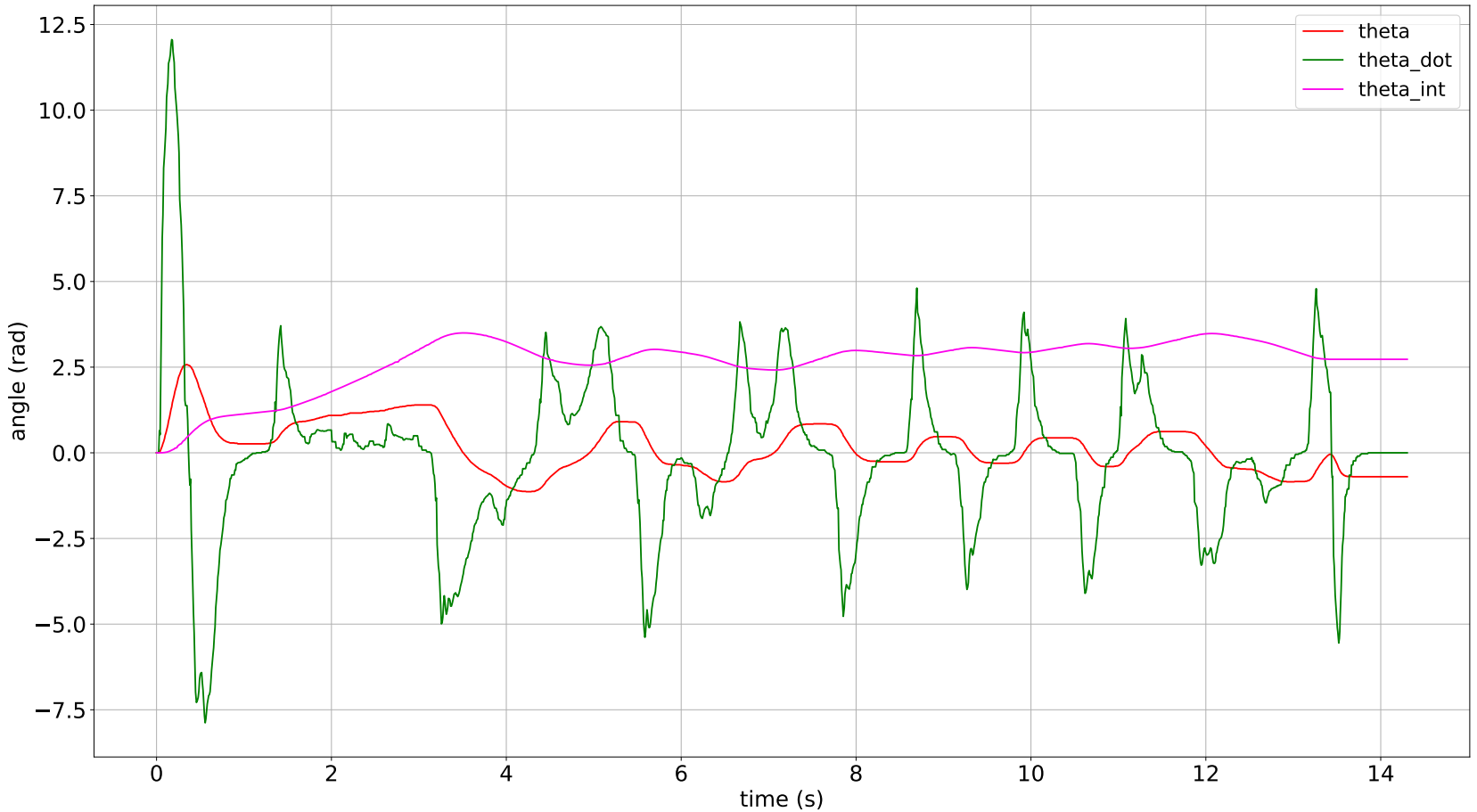




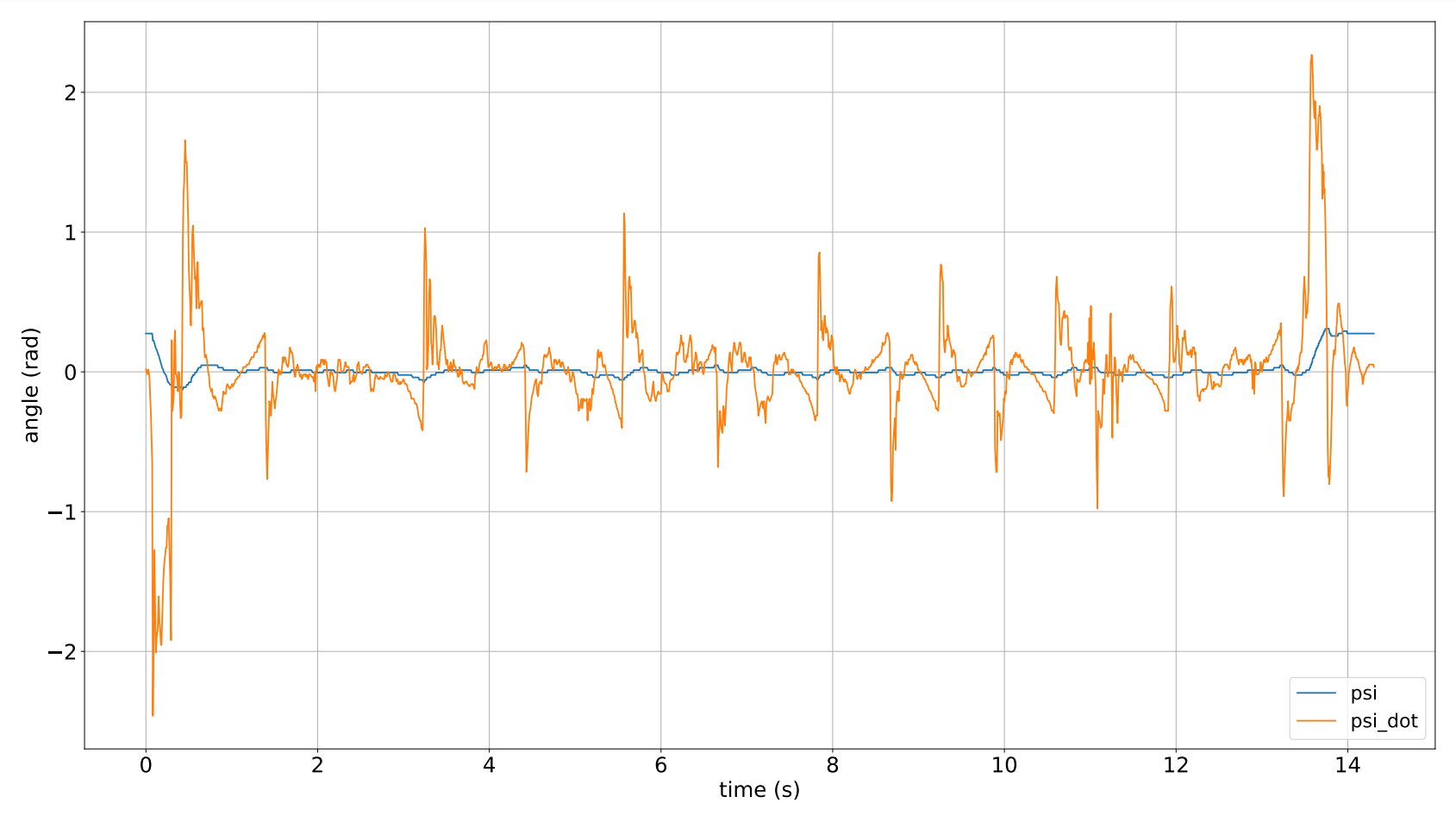
C++ IMPLEMENTATION

- Takes advantage of C++ notoriously good performance even on embedded systems
- Cross-compiled through a Docker image provided by the ev3dev project
- Exploits Linux system calls to give higher priority to both the control loop thread and the whole process
- Exceptional performance with an average of 5ms and a 95% quantile of 3ms

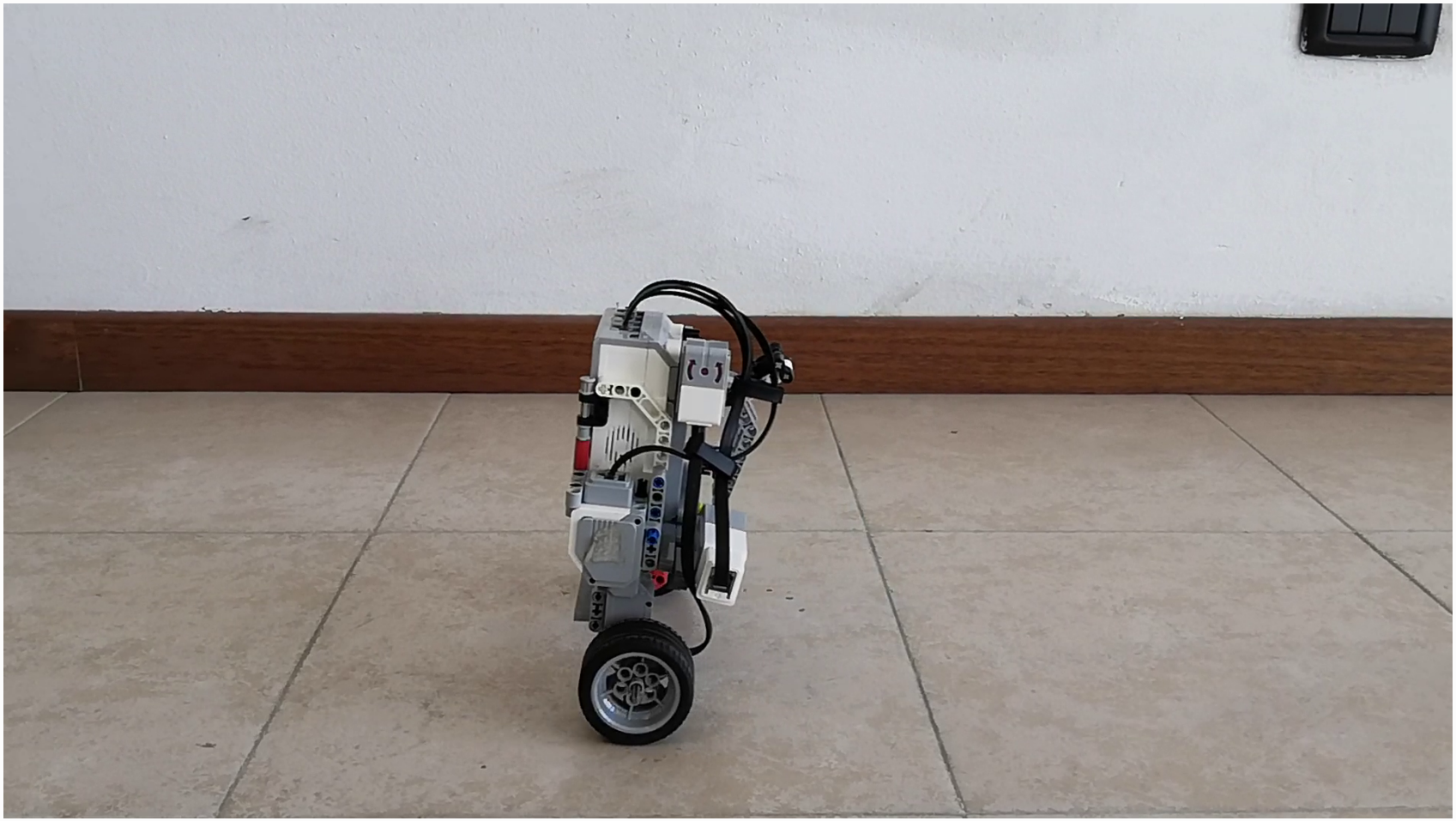
C++ IMPLEMENTATION
PLOT



C++ IMPLEMENTATION
PLOT



Video demo



Robot movement

Model extension

- We extend the model to deal with a variable $\dot{\theta}$ reference, but with the references for ψ and $\dot{\psi}$ equal to zero.
- The state of the new model becomes:

$$\tilde{x}(n) = \begin{bmatrix} \theta_{err}(n) & \psi(n) & \dot{\theta}_{err}(n) & \dot{\psi}(n) & \tilde{\theta}_{int}(n) \end{bmatrix}^T$$

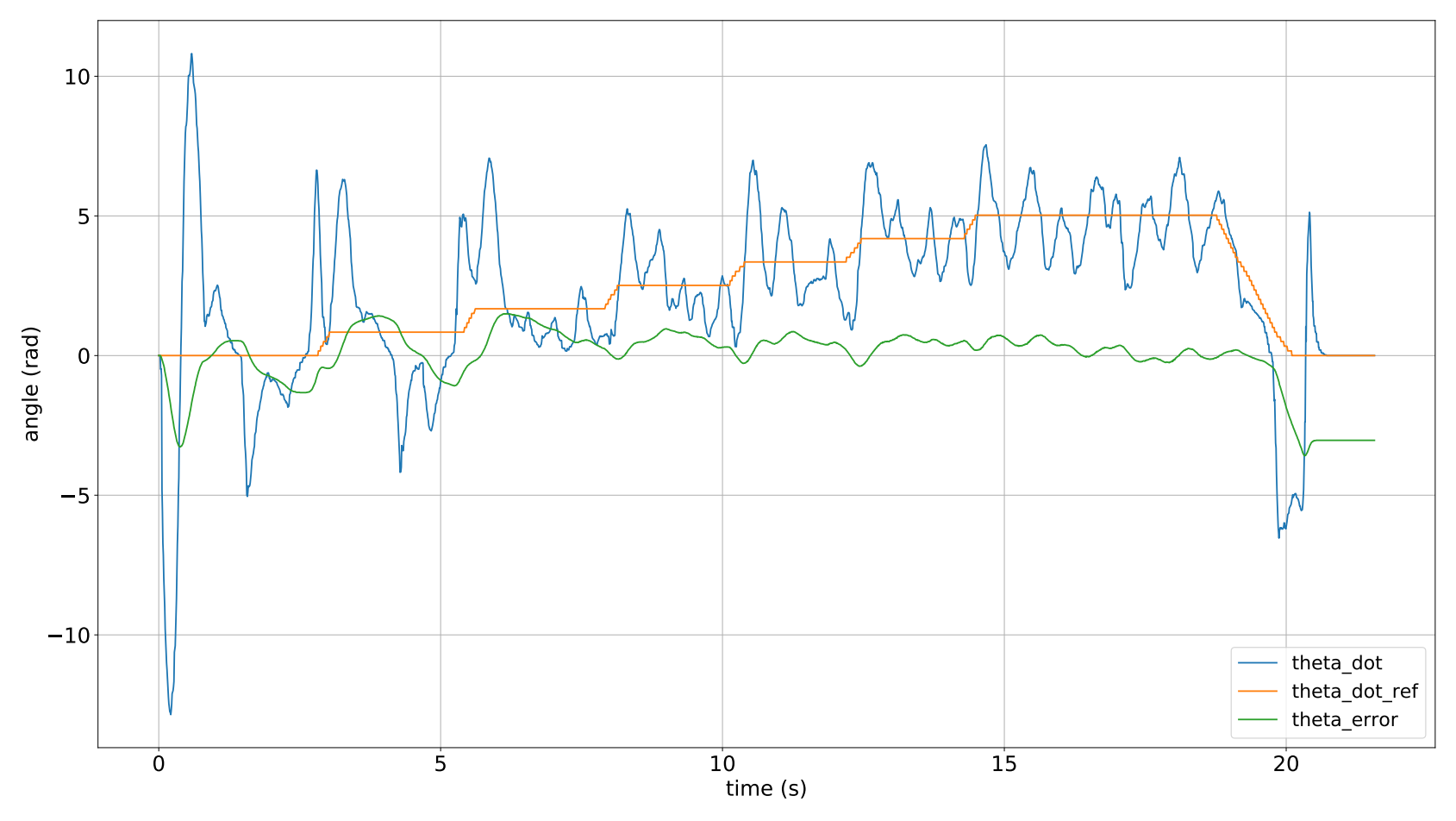
  - $\dot{\theta}_{err}(n) = \dot{\theta}(n) - \dot{\theta}_{ref}(n)$
  - $\theta_{err}(n) = \theta(n) - \theta_{ref}(n)$, with
    - $\theta_{ref}(n) = \theta_{ref}(n-1) + [t(n) - t(n-1)] \cdot \dot{\theta}_{ref}(n)$
  - $\tilde{\theta}_{int}(n) = \tilde{\theta}_{int}(n-1) + [t(n) - t(n-1)] \cdot \theta_{err}(n-1)$
- We add the possibility of controlling the robot steering through a steering component s_{ref}:

$$u = \begin{bmatrix} V_l \mp s_{ref} & V_r \pm s_{ref} \end{bmatrix}^T$$

Model extension

Plot

- To avoid abrupt changes in the reference speed we also apply a variable number of stabilization cycles which gradually adjust the speed



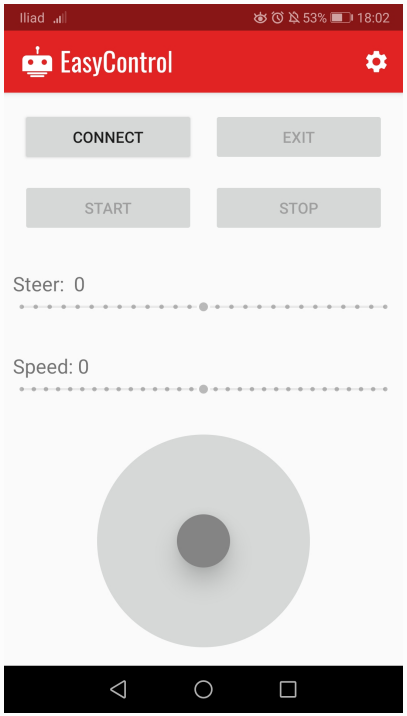
Remote Control
SSH

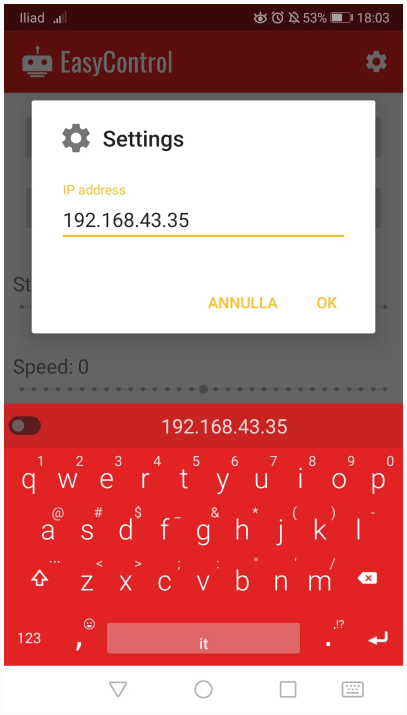
- C++ implementation extended to accept user input to set speed and steering reference values
- Text commands, e.g. `speed_ref=10`, `steer_ref=-1`
- Bounds on maximum and minimum values for user input reference in order not to make the robot unstable

```
~/CLionProjects/segway   robot   ssh robot@192.168.43.35
             _____       _
  _____   _|___ /  __| | _____   __
 / _ \ \ / / |_ \ / _` |/ _ \ \ / /
|  __/\ V / ___) | (_| |  __/\ V /
 \___| \_/ |____/ \__,_|\___| \_/

Debian stretch on LEGO MINDSTORMS EV3!
robot@ev3dev:~$ sudo ./segway
[sudo] password for robot:
Connected

start
Calibrating... calibration finished
Starting control loop...
Give commands (speed_ref=, steer_ref=) to control the robot.

speed_ref=10
steer_ref=1
stop
exit

Ended
---
robot@ev3dev:~$ _
```

Remote Control
EasyControl App

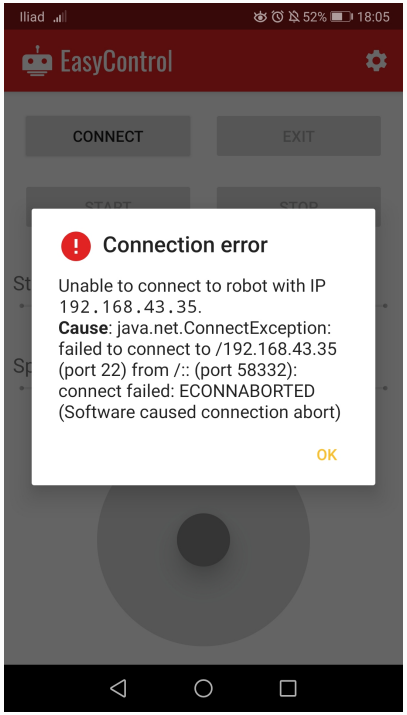
- Implemented with Kotlin programming language and Android Studio IDE
- Easy to use interface, robot and smartphone only need to be connected to the same Wi-Fi network
- The user can customize the IP address of the robot it connects to

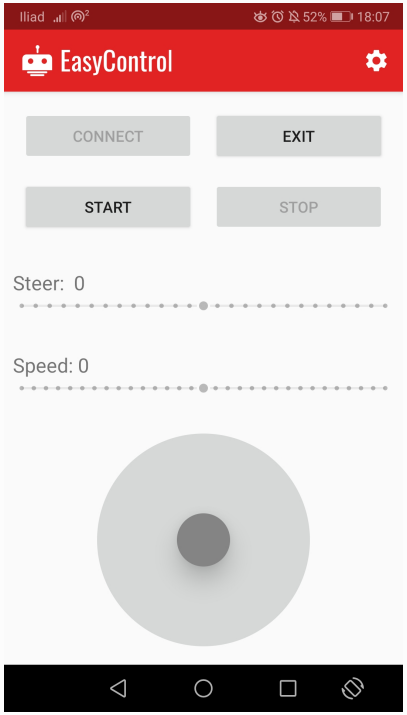


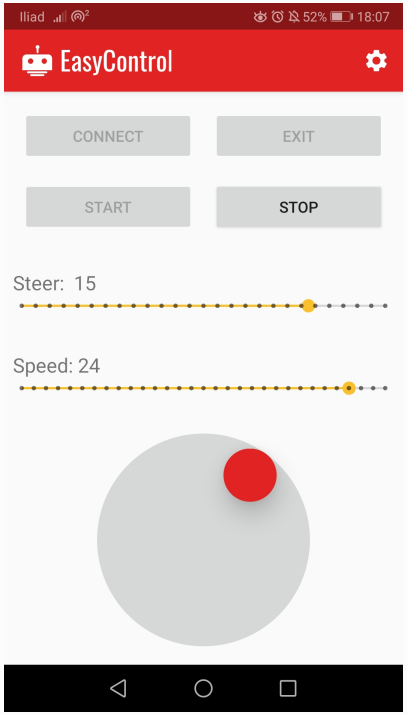


Remote Control
EasyControl App

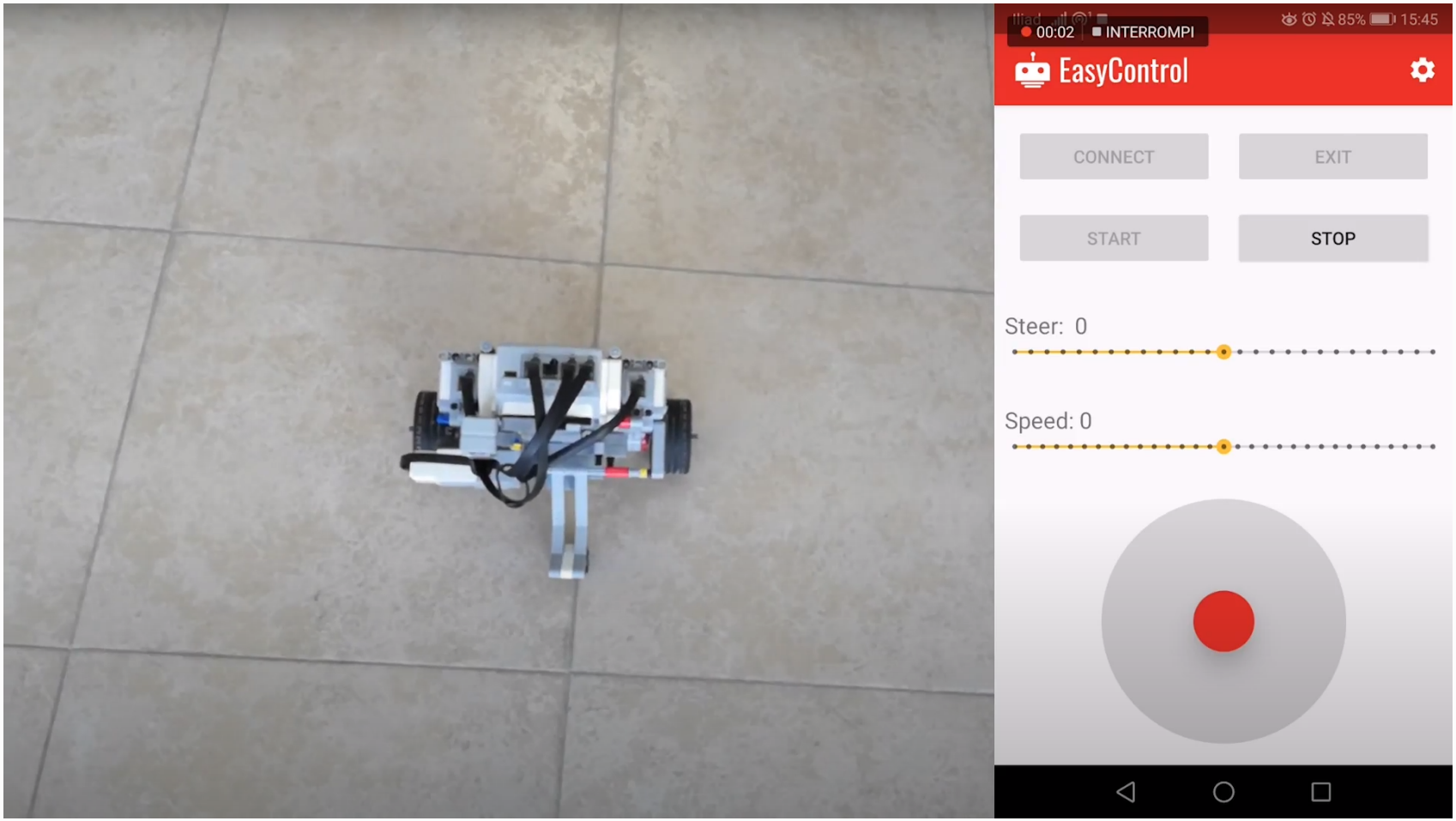
- The app shows only the options available to the user
- The app sends command to the robot at 60Hz (every 16ms)







Video demo


00:02
INTERROMPI
85%
15:45
EasyControl
CONNECT
EXIT
START
STOP
Steer: 0
Speed: 0

Conclusions

Conclusions

- We described various implementations of a control system for a Segway-like two-wheeled robot using ev3dev
- Both Python implementations did not manage to stabilize the robot
- The C++ implementation proved to be performant enough to stabilize the robot
- We achieved full movement in 2D by extending the original model
- We developed an app to allow easy and user-friendly remote control experience
- Future work: equip the robot with a proximity sensor for obstacle detection and avoidance