OpenBot: Turning Smartphones into Robots

Matthias Müller
Intel Labs

Vladlen Koltun
Intel Labs

Abstract: Current robots are either expensive or make significant compromises on sensory richness, computational power, and communication capabilities. We propose to leverage smartphones to equip robots with extensive sensor suites, powerful computational abilities, state-of-the-art communication channels, and access to a thriving software ecosystem. We design a small electric vehicle that costs $50 and serves as a robot body for standard Android smartphones. We develop a software stack that allows smartphones to use this body for mobile operation and demonstrate that the system is sufficiently powerful to support advanced robotics workloads such as person following and real-time autonomous navigation in unstructured environments. Controlled experiments demonstrate that the presented approach is robust across different smartphones and robot bodies.

Keywords: mobile robots, smartphones, autonomous navigation, learning

1 Introduction

Robots are expensive. Legged robots and industrial manipulators cost as much as luxury cars, and even the cheapest robots from Franka Emika or Clearpath cost at least $10K. Few academic labs can afford to experiment with robotics on the scale of tens or hundreds of robots.

A number of recent efforts have proposed designs for more affordable robots. Kau *et al.* [20] and Grimminger *et al.* [14] proposed quadruped robots that rely on low-cost actuators and cost $3K and €4K. Yang *et al.* [40], Gupta *et al.* [15], and Gealy *et al.* [11] proposed manipulation robots that cost $2K, $3K, and $5K respectively. A number of mobile robots for hobbyist and researchers have been released which fall in the $250–500 range. These include the AWS DeepRacer [1], the DJI Robomaster S1 [9], the Nvidia JetBot [24], and the DukieBot [28]. In order to achieve this price point, these platforms had to make compromises with regards to the physical body, sensing, communication, and compute. Is there an alternative where robots become extremely cheap, accessible to everyone, and yet possess extensive sensory abilities and computational power?

In this work, we push further along the path to highly capable mobile robots that could be deployed at scale. Our key idea is to leverage smartphones. We are inspired in part by projects such as Google Cardboard: by plugging standard smartphones into cheap physical enclosures, these designs enabled millions of people to experience virtual reality for the first time. Can smartphones play a similar role in robotics?

More than 40% of the world's population own smartphones. Commodity models now carry HD cameras, powerful CPUs and GPUs, advanced IMUs, GPS, WiFi, Bluetooth, 4G modems, and more. Modern smartphones are even equipped with dedicated AI chips for neural network inference, some of which already outperform common desktop processors [18].

We develop and validate a design for a mobile robot that leverages a commodity smartphone for sensing and computation (Figure 1). The smartphone acts as the robot's brain and sensory system. This brain is plugged into a cheap electromechanical body that costs less than $50.

Using off-the-shelf smartphones as robot brains has numerous advantages beyond cost. Hardware components on custom robots are quickly outdated. In contrast, consumer-grade smartphones undergo generational renewal on an annual cadence, acquiring higher-resolution and higher-framerate cameras, faster processors, new sensors, and new communication interfaces. As a side effect, second-hand smartphones are sold cheaply, ready for a second life as a robot. In addition to the rapid advancement of hardware capabilities, smartphones benefit from a vibrant software ecosys-

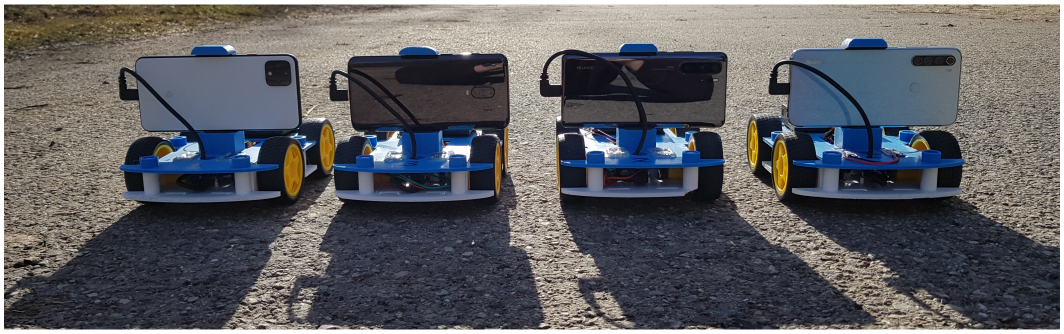

Figure 1: **OpenBots.** Our wheeled robots leverage a smartphone for sensing and computation. The robot body costs $50 without the smartphone. The platform supports person following and real-time autonomous navigation in unstructured environments.

tem. Our work augments this highly capable bundle of sensing and computation with a mobile physical body and a software stack that supports robotics workloads.

Our work makes four contributions. (1) We design a small electric vehicle that relies on cheap and readily available components with a hardware cost of only $50 as a basis for a low-cost wheeled robot. (2) We develop a software stack that allows smartphones to use this vehicle as a body, enabling mobile navigation with real-time onboard sensing and computation. (3) We show that the proposed system is sufficiently powerful to support advanced robotics workloads such as person following and autonomous navigation. (4) We perform extensive experiments that indicate that the presented approach is robust to variability across smartphones and robot bodies.

Our complete design and implementation, including all hardware blueprints and the software suite will be made freely available to support affordable robotics research and education at scale.

2 Related Work

Wheeled robots used for research can be divided into three main categories: tiny robots used for swarm robotics, larger robots based on RC trucks used for tasks that require extensive computation and sensing, and educational robots. Swarm robots [3, 33, 23, 38] are inexpensive but have very limited sensing and compute. They are designed to operate in constrained indoor environments with emphasis on distributed control and swarm behaviour. On the other end of the spectrum are custom robots based on RC trucks [13, 6, 36, 25, 1, 12]. They feature an abundance of sensors and computation, supporting research on problems such as autonomous navigation and mapping. However, they are expensive and much more difficult to assemble and operate. Educational robots [29, 28] are designed to be simple to build and operate while maintaining sufficient sensing and computation to showcase some robotic applications such as lane following. However, their sensors and compute are usually not sufficient for cutting-edge research. Some robots such as the DuckieBot [28] and Jetbot [24] try to bridge this gap with designs that cost roughly $250. However, these vehicles are small and slow. In contrast, our wheeled robot body costs $50 or less and has a much more powerful battery, bigger chassis, and four rather than two motors. The body serves as a plug-in carrier for a smartphone, which provides computation, sensing, and communication. Leveraging off-the-shelf smartphones allows this design to exceed the capabilities of much more expensive robots.

Contemporary smartphones are equipped with mobile AI accelerators, the capabilities of which are rapidly advancing. Ignatov *et al.* [18] benchmark smartphones with state-of-the-art neural networks for image classification, image segmentation, object recognition, and other demanding workloads. Not only are most recent smartphones able to run these complex AI models, but they approach the performance of CUDA-compatible graphics cards. Lee *et al.* [21] show how to leverage mobile GPUs that are already available on most smartphones in order to run complex AI models in real time. They also discuss design considerations for optimizing neural networks for deployment on smartphones. Our work harnesses these consumer hardware trends for robotics.

There have been a number of efforts to combine smartphones and robotics. In several hobby projects, smartphones are used as a remote control for a robot [27, 32]. On Kickstarter, Botiful [8] and Romo [30] raised funding for wheeled robots with smartphones attached for telepresence, and

Ethos [39] for a drone powered by a smartphone. Most related to our work is Wheelphone [10], where a smartphone is mounted on a robot for autonomous navigation. Unfortunately, this project is stale; the associated Github repos have only 1 and 4 stars and the latest contribution was several years ago. The robot has only two motors, a maximum speed of 30 cm/s, and is restricted to simple tasks such as following a black tape on the floor or obstacle avoidance on a tabletop. Despite these drawbacks, it costs $250. Our robot is more rugged, can reach a maximum speed of 150 cm/s, costs $35-50, and is capable of heavy robotic workloads such as autonomous navigation.

Researchers have also explored the intersection of smartphones and robotics. Yim *et al.* [41] detect facial expressions and body gestures using a smartphone mounted on a robot to study social interaction for remote communication via a robotic user interface. DragonBot [35] is a cloud-connected 5-DoF toy robot for studying human/robot interaction; a smartphone is used for control and a visual interface. V.Ra [4] is a visual and spatial programming system for robots in the IoT context. Humans can specify a desired trajectory via an AR-SLAM device (*e.g.* a smartphone) which is then attached to a robot to execute this trajectory. In contrast to our work, the navigation is not autonomous but relies on user input. Oros *et al.* [26] leverage a smartphone as a sensor suite for a wheeled robot. The authors retrofit an RC truck with a smartphone mount and I/O interface to enable autonomous operation. However, the key difference is that they stream the data back to a computer for processing. Moreover, the proposed robot costs $350 without the smartphone and does not leverage recent advancements that enable onboard deep neural network inference or visual-inertial state estimation. The project is stale, with no updates to the software in 7 years.

In summary, the aforementioned projects use the smartphone as a remote control for teleoperation, offload data to a server for processing, or rely on commercial or outdated hardware and software. In contrast, our platform turns a smartphone into the brain of a fully autonomous robot with onboard sensing and computation.

3 System

3.1 A Body for a Low-cost Wheeled Robot

A brain without a body cannot act. In order to leverage the computational power and sensing capabilities of a smartphone, the robot needs an actuated body. We develop a body for a low-cost wheeled robot which only relies on readily available electronics and 3D printing for its chassis. The total cost is $50 for building a single body, with 40% of that cost due to good batteries. If building multiple robots, the cost further decreases, for example by 30% for 5 units. Table 1 shows the bill of materials. In the following, we discuss the mechanical and electrical design in more detail.

Component	**Quantity**	**Unit Price**	**Bulk Price**
3D-printed Body	1	$5.00	$5.00
Speed Sensor	2	$2.00	$1.00
Motor + Tire	4	$3.50	$2.00
Motor Driver	1	$3.00	$2.50
Arduino Nano	1	$8.00	$3.50
Battery 18650	3	$7.00	$4.00
Miscellaneous	1	$5.00	$2.00
Total		**$50**	**$35**

Table 1: **Bill of materials.** Unit price is the approximate price per item for a single vehicle. The bulk price is the approximate price per item for five vehicles.

Mechanical design. The chassis of the robot is 3D-printed and is illustrated in Figure 2. It consists of a bottom plate and a top cover with LEGO-like connectors that click into place. The bottom plate features the mounting points for the motors and electronics. The four motors are fixed with eight M3x25 screws. The motor controller and microcontroller attach to the bottom plate. There are appropriate openings for the indicator LEDs and grooves for the encoder disks mounted on the front wheels. The top plate features a universal smartphone mount which uses two springs to adjust to different phones. There is also an opening for the USB cable that connects the smartphone to an Arduino microcontroller and grooves for the optical wheel odometry sensors.

With standard settings on a consumer 3D printer (*e.g.* Ultimaker S5), the complete print requires 13.5 hours for the bottom plate and 9.5 hours for the top cover with the phone mount. It is possible to print at a faster pace with less accuracy. The material weight is 146g for the bottom and 103g for the top. Considering an average PLA filament price of $20/kg, the total material cost is about $5.

Electrical design. The electrical design is shown in Figure 3. We use the LM298N breakout board as motor controller. The two left motors are connected to one output and the two right motors to the other. The battery pack is connected to power terminals to provide power to the motors as needed.

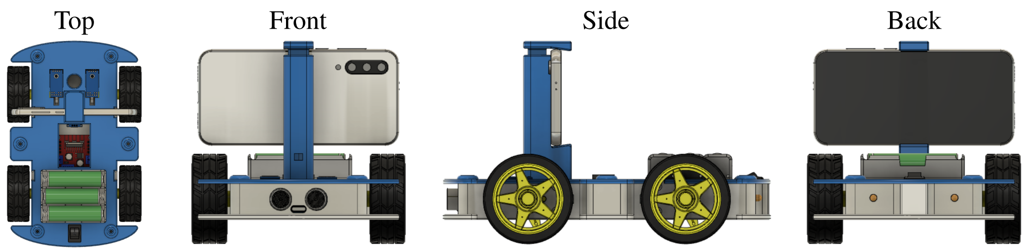

Figure 2: **Mechanical design.** CAD design of the 3D-printed robot body.

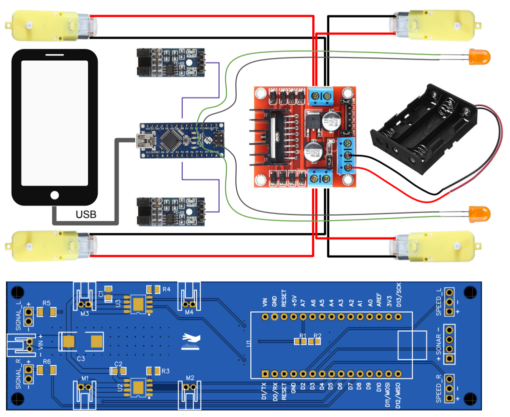

Figure 3: **Connection diagram.** *Top:* Electrical connections between battery, motor controller, microcontroller, speed sensors, indicator LEDs, and smartphone. *Bottom:* Optional custom PCB to reduce wiring.

Our battery consists of three USB-rechargeable 18650 Lithium cells connected in series, providing a voltage between 9.6V and 12.6V depending on their state-of-charge (SOC). An Arduino Nano board is connected to the smartphone via its USB port, providing a serial communication link and power. Two LM393-based speed sensors with optical sensors are connected as input to two of the digital pins. The two front wheels are each equipped with a disk that interrupts the optical signal: these interruptions are detected and counted by the Arduino, providing a wheel odometry signal. Two further digital pins are used as outputs to switch the indicator LEDs on and off, providing visual means for the robot to communicate with its environment. We also use one of the analog pins as input to measure the battery voltage through a voltage divider. Finally, four PWM pins are connected to the motor controller. This allows us to adjust the speed and direction of the motors according to the control commands received from the smartphone. We have also designed a PCB with integrated battery monitoring and two TI-DRV8871 motor drivers for increased efficiency. The Arduino, motors, indicator LEDs, speed sensors, and an optional ultrasonic sensor are simply plugged in. When building multiple robots, the PCB further reduces setup time and cost.

3.2 Software Stack

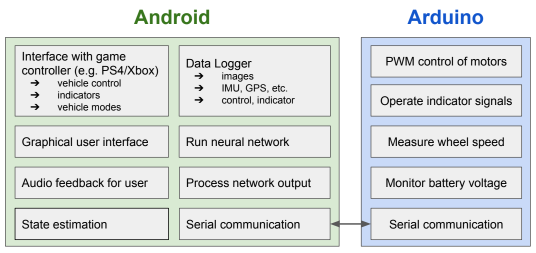

Figure 4: **Software design.** Our Android application is responsible for high-level computation on the smartphone and the Arduino program provides the low-level interface to the vehicle.

Our software stack consists of two components, illustrated in Figure 4. The first is an Android application that runs on the smartphone. Its purpose is to provide an interface for the operator, collect datasets, and run the higher-level perception and control workloads. The second component is a program that runs on the Arduino. It takes care of the low-level actuation and some measurements such as wheel odometry and battery voltage. The Android application and the Arduino communicate via a serial communication link. In the following, we discuss both components in more detail.

Android application. We design a user interface which provides visual and auditory feedback for interaction with the robot. We use Bluetooth communication to interface with common game console controllers (*e.g.* PS4, Xbox), which can be used to teleoperate the robot for data collection. (The use of a controller is optional and is not needed for autonomous operation.) To collect data, such as demonstrations for imitation learning, we use the joystick inputs to control the robot and use the buttons to trigger functionalities such as toggling control modes, logging, running a neural network, *etc.* We derive our graphical user interface from the Android Tensorflow Object Detection application [37] and extend it. Our GUI provides the camera feed and buttons to toggle data logging, control modes, and serial communication. It also allows switching between different neural networks to control the vehicle and provides relevant information such as image resolution, inference time and predicted controls. We also integrate voice feedback for operation via the game controller.

The Android ecosystem provides a unified interface to obtain sensor data from any Android smartphone. We build a data logger on top of that in order to collect datasets with the robots. Currently, we record readings from the following sensors: camera, gyroscope, accelerometer, magnetometer, ambient light sensor, and barometer. Using the Android API, we are able to obtain the following sensor readings: RGB images, angular speed, linear acceleration, gravity, magnetic field strength, light intensity, atmospheric pressure, latitude, longitude, altitude, bearing, and speed. In addition to the phone sensors, we also record body sensor readings (wheel odometry and battery voltage), which are transmitted via the serial link.

We leverage the computational power of the smartphone to process the sensory input and compute the robots' actions in real time. While there are many classic motion planning algorithms, we focus on learning-based approaches, which allow for a unified interface. In particular, we rely on the Tensorflow Lite infrastructure, which integrates seamlessly with smartphones [18, 21]. Our Android application features model definitions for object detection and autonomous navigation. These define the input and output properties of the neural network. We build on top of the Tensorflow Object Detection application [37] to detect people and perform visual servoing to follow them. We also integrate a model for autonomous navigation inspired by Conditional Imitation Learning [6]. The deployment process is simple. After training a model in Tensorflow, it is converted to a Tensorflow Lite model that can be directly deployed on the smartphone.

Arduino program. We use an Arduino Nano microcontroller to act as a bridge between the vehicle body and the smartphone. Its main task is to handle the low-level control of the vehicle and provide readings from low-level vehicle-mounted sensors. The program components are shown on the right in Figure 4. The Arduino receives the vehicle controls and indicator signals via the serial connection. It converts the controls to PWM signals for the motor controller and toggles the LEDs according to the indicator signal. The Arduino program also keeps track of the wheel rotations by counting the interrupts of optical sensors on the left and right front wheels. It calculates the battery voltage by a scaled moving average of measurements at the voltage divider circuit. These measurements are sent back to the Android application through the serial link.

Platform	Retail Cost	Setup Time [h]	Size [cm]	Weight [kg]	Speed [m/s]	Battery [min]	Actuation	Odometry	Camera	LiDAR	IMU	GPS	WiFi	Bluetooth	3G/4G/5G	Speaker	Microphone	Display	Compute	Ecosystem
AutoRally [12]	$10,000	100	100x60x40	22	25	20+	BLDC+Servo	✓	✓	✗	✓	✓	✓	✗	✗	✗	✗	✗	Mini-ITX PC	ROS
F1/10 [25]	$3600	3	55x30x20	4.5	18	20+	BLDC+Servo	✓	✓	✓	✓	✗	✓	✓	✗	✗	✗	✗	Jetson TX2	ROS
RACECAR [19]	$3400	10	55x30x20	4.5	18	20+	BLDC+Servo	✓	✓	✓	✓	✗	✓	✓	✗	✗	✗	✗	Jetson TX1	ROS
BARC [13]	$1000	3	54x28x21	3.2	-	20+	BLDC+Servo	✓	✓	✗	✓	✗	✓	✗	✗	✗	✗	✗	Odroid XU-4	ROS
MuSHR [36]	$900	3	44x28x14	3	11	20+	BLDC+Servo	✓	✓	✓	✓	✗	✓	✓	✗	✗	✗	✗	Jetson Nano	ROS
DeepRacer [1]	$400	0.25	-	-	6	15+	BDC+Servo	✗	✓	✗	✓	✗	✓	✗	✗	✗	✗	✗	Intel Atom	Custom
DonkeyCar [31]	$250	2	25x22x12	1.5	9	15+	BDC+Servo	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	Raspberry Pi	Custom
Duckiebot [28]	$280	0.5	-	-	-	-	2xBDC	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	Raspberry Pi	Custom
Pheeno [38]	$270	-	13x11	-	0.42	300+	2xBDC	✓	✗	✗	✓	✗	✓	✓	✗	✗	✗	✗	ARM Cortex-A7	Custom
JetBot [24]	$250	1	20x13x13	-	-	-	2xBDC	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	Nvidia Jetson	Custom
Create-2 [7]	$200	-	34x34x9	3.6	-	-	2xBDC	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	Custom
Thymio II [29]	$170	-	11x11x5	0.46	0.14	-	2xBDC	✗	✗	✗	✓	✗	✗	✗	✗	✓	✓	✗	Microcontroller	Custom
AERobot [34]	$20	0.1	3x3x3	0.03	-	-	2xVibration	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	Microcontroller	Custom
OpenBot	$50⋆	0.5	24x15x12	0.7	1.5	45+	4xBDC	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	Smartphone	Android

Table 2: **Robots.** Comparison of wheeled robotic platforms. Top: Robots based on RC trucks. Bottom: Navigation robots for deployment at scale and in education. "–" indicates that no information is available. ⋆ The cost of the smartphone is not included and varies.

3.3 Comparison to Other Wheeled Robots

We compare to existing robot platforms in Table 2. In contrast to other robots, our platform has an abundance of processing power, communication interfaces, and sensors provided by the smartphone. Existing robots often rely on custom software ecosystems, which require dedicated lab personnel who maintain the code, implement new features, and implement drivers for new sensors. In contrast, we use Android, one of the largest constantly evolving software ecosystems. All the low-level software for sensor integration and processing already exists and improves without any additional effort by the robotics community. All sensors are already synchronized on the same clock, obviating what is now a major challenge for many existing robots.

4 Validation

This section outlines the experiments we conduct with the presented platform. We validate that smartphones are suitable to provide sensing, communication, and compute for interesting robotics applications. We first discuss our general evaluation setup and procedure that ensures a fair comparison. Then, we present the experimental setup for two applications, person following and autonomous navigation. The results are reported in Section 5.

4.1 Evaluation

Smartphones. We validate the presented approach with a variety of popular smartphones from the past two years with prices ranging from \$120 to \$750. The smartphones are carefully selected to cover different manufactures, chipsets, and sensor suites. Detailed specifications and benchmark scores of the smartphones are provided in the Appendix B.

Evaluation metrics. In order to streamline our evaluation while providing a comprehensive performance summary, we use three metrics: distance, success rate, and collisions. The distance is continuous and we report it as a percentage of the complete trajectory. The distance measurement stops if an intersection is missed, a collision occurs or the goal is reached. The success rate is binary and indicates whether or not the goal was reached. We also count the number of collisions. All results are averaged across three trials.

Evaluation protocol. Since our experiments involve different smartphones, cheap robots, and a dynamic physical world, we make several considerations to ensure a fair evaluation. We divide each experiment into several well-defined segments to ensure consistency and minimize human error. To ensure that the robots are initialized at the same position for each experiment, we use markers at the start and end position of each segment. We also align all phones with their power button to the phone mount to ensure the same mounting position across experiments. Since the inference time of smartphones can be affected by CPU throttling, we check the temperature of each smartphone before starting an experiment and close all applications running in the background. We use several metrics to provide a comprehensive performance analysis.

4.2 Person Following

Network. We use the SSD object detector with a pretrained MobileNet backbone [17]. To investigate the impact of inference time, we use two different versions of MobileNet, the original MobileNetV1 [17] and the lastest MobileNetV3 [16]. We use the pretrained models released as part of the Tensorflow object detection API. Both models were trained on the COCO dataset [22] with 90 class labels. The models are quantized for improved inference speed on smartphone CPUs.

Experimental setup. We only consider detections of the person class and reject detections with a confidence below a threshold of 50%. We track detections across frames, and pick the one with the highest confidence as the target. We apply visual servoing with respect to the center of the bounding box, keeping the person centered in the frame. We evaluate both the MobileNetV1 and MobileNetV3 variants of the object detector across ten different smartphones. For fair comparison, we only use the CPU with one thread on each phone. Using the GPU or the NNAPI can further improve the runtime on most phones. We provide a quantitative evaluation in a controlled indoor environment. The route involves a round trip between an office and a coffee machine and includes four left turns and four right turns. We average results across three trials for each experiment. In addition, the supplementary video contains qualitative results in unstructured outdoor environments.

4.3 Autonomous Navigation

Network. We design a neural network similar in spirit to the *command-input* variant of Conditional Imitation Learning [6]. Our network is about one order of magnitude smaller than existing networks and is able to run in real time even on mid-range smartphones. We train this network using a novel loss function and validation metrics. We obtain successful navigation policies with less than 30 minutes of labeled data and augmentation. The network architecture, dataset acquisition, loss function, training details and validation metrics are further discussed in Appendix C.

Experimental setup. The robots have to autonomously navigate through corridors in an office building without colliding. The driving policy receives high-level guidance in the form of indicator commands such as *turn left / right at the next intersection* [6]. Each trial consists of several segments with a total of 2 straights, 2 left turns, and 2 right turns. More details on the setup including a map with dimensions are provided in Appendix C.

5 Results

5.1 Person Following

In this experiment, we investigate the feasibility of running complex AI models on smartphones. We use object detectors and apply visual servoing to follow a person. Our experiments show that all recent mid-range smartphones are able to track a person consistently at speeds of 10 fps or higher. The cheapest low-end phone (Nokia 2.2) performs worst, but is surprisingly still able to follow the person about half of the time. We expect that even low-end phones will be able to run complex AI models reliably in the near future. The Huawei P30 Pro was the best performer in our comparison, closely followed by other high-end phones such as the Google Pixel 4XL and the Xiaomi Mi9. All recent mid-range phones (*e.g.* Xiaomi Note 8, Huawei P30 Lite, Xiaomi Poco F1) clearly outperform the Samsung Galaxy Note 8, which was a high-end phone just two years ago. This is due to dedicated AI accelerators present in recent smartphones [18] and highlights the rapid rate at which smartphones are improving. Please see the supplementary video for qualitative results.

	Distance ↑		Success ↑		Collisions ↓		FPS ↑	
MobileNet	V1	V3	V1	V3	V1	V3	V1	V3
Huawei P30 Pro	100%	100%	100%	100%	0.0	0.0	33	30
Google Pixel 4XL	100%	100%	100%	100%	0.0	0.0	32	28
Xiaomi Mi9	100%	100%	100%	100%	0.0	0.0	32	28
Samsung Note 10	100%	100%	100%	100%	0.0	0.0	16	22
OnePlus 6	100%	100%	100%	100%	0.0	0.0	11	15
Huawei P30 Lite	100%	99%	100%	83%	0.0	0.3	9	11
Xiaomi Note 8	100%	100%	100%	100%	0.0	0.0	9	11
Xiaomi Poco F1	98%	100%	83%	100%	0.3	0.0	8	12
Samsung Note 8	58%	100%	33%	100%	0.0	0.0	6	10
Nokia 2.2	37%	50%	0%	0%	0.0	0.3	4	5

Table 3: **Person following.** We use MobileNet detectors and visual servoing to follow a person. All results are averaged across three trials.

5.2 Autonomous Navigation

We train a driving policy that runs in real time on most smartphones. Our learned policy is able to consistently follow along corridors and take turns at intersections. We compare it to existing driving policies and achieve similar performance as the baselines while requiring about one order of magnitude fewer parameters. We also successfully transfer our driving policy to different smartphones and robot bodies. When training on data acquired with multiple smartphones and robots, we observe increased robustness. We show that our driving policy is able to generalize to previously unseen environments, novel objects, and even dynamic obstacles such as people even though only static obstacles were present in the training data.

Comparing driving policies. OpenBot enables benchmarking using real robots. We compare our policy to two baselines across three trials in Table 4. To ensure optimal conditions for the baselines, we use the high-end smartphone Xiaomi Mi9. Our driving policy network is smaller by a factor of 7 or more than the baselines. Yet it outperforms PilotNet [2] and achieves similar performance to CIL [6] while running at twice the speed.

	Distance ↑	Success ↑	Collisions ↓	FPS ↑	Params ↓
PilotNet [2]	92±0%	83±0%	0.0±0.0	60±1	9.6M
CIL [6]	94±5%	89±10%	0.0±0.0	20±1	10.7M
Ours	94±5%	89±10%	0.0±0.0	47±2	1.3M

Table 4: **Baselines.** We compare our driving policy to two baselines. All policies are trained for 100 epochs using the same data and hyperparameters.

Generalization to novel phones. Table 5 shows that our navigation policy can be trained with data from one phone and generalize to other phones. How well the generalization works depends on the target phone, especially its processing power and camera placement. We observe a degradation in performance for phones unable to run the driving policy in real time. Differences in camera placement affect qualitative driving performance; for tasks that require high precision this may need to be accounted for. The differences in camera sensors (*e.g.* color reproduction and exposure) are largely overcome by data augmentation.

	Distance ↑	Success ↑	Collisions ↓	FPS ↑
Xiaomi Mi9	94±5%	89±10%	0.0±0.0	47±2
Google Pixel 4XL	92±0%	83±0%	0.0±0.0	57±3
Huawei P30 Pro	97±5%	94±10%	0.0±0.0	51±0
Samsung Note 10	92±0%	83±0%	0.0±0.0	38±8
OnePlus 6T	89±5%	78±10%	0.1±0.1	32±0
Xiaomi Note 8	92±0%	83±0%	0.0±0.0	31±0
Huawei P30 Lite	92±0%	83±0%	0.0±0.0	30±1
Xiaomi Poco F1	86±5%	72±10%	0.1±0.1	26±8
Samsung Note 8	83±0%	67±0%	0.2±0.0	19±3

Table 5: **Novel phones.** We train our driving policy using one phone (top) and then test it on other phones (bottom).

Generalization to novel bodies. Table 6 shows that our navigation policy can be trained with data from one robot body and generalize to other robot bodies. Due to the cheap components, every body exhibits different actuation noise which may change over time and is observable in its behaviour (*e.g.* a tendency to pull to the left or to the right). We address this by injecting noise in the training process [6]. Further details are provided in Appendix D.

	Distance ↑	Success ↑	Collisions ↓
Robot Body 1	94±5%	89±10%	0.0±0.0
Robot Body 2	94±5%	89±10%	0.0±0.0
Robot Body 3	92±0%	83±0%	0.0±0.0
Robot Body 4	89±5%	78±10%	0.1±0.1

Table 6: **Novel bodies.** We train our driving policy using one body (top) and then test it on other bodies (bottom).

Generalization to novel obstacles. Even though our driving policies were only exposed to static obstacles in the form of office chairs during data collection, they were able to generalize to novel static obstacles (potted plants) and even dynamic obstacles (people) at test time. The low image resolution, aggressive downsampling, and small number of parameters in our network may serve as natural regularization that prevents the network from overfitting to specific obstacles. Since the network processes camera input on a frame-by-frame basis, static and dynamic obstacles are treated on the same basis. We also conjecture that the network has learned some robustness to motion blur due to vibrations of the vehicle. Our navigation policy is also able to generalize to novel environments within the same office building. Please refer to the supplementary video for qualitative results.

Learning with multiple robots. We also investigated the impact of using multiple different smartphones and robot bodies for data collection which is relevant for using our platform at scale. We provide detailed results in Appendix D and summarize the findings here. Training the driving policies on data acquired from multiple smartphones improves generalization to other phones; every manufacturer tunes the color reproduction and exposure slightly differently, leading to natural data augmentation. The driving policy trained on data acquired with multiple robot bodies is the most robust; since the smartphone was fixed, the network had to learn to cope with noisy actuation and dynamics, which we show to be possible even with relatively small datasets.

6 Conclusion

This work aims to address two key challenges in robotics: accessibility and scalability. Smartphones are ubiquitous and are becoming more powerful by the year. We have developed a combination of hardware and software that turns smartphones into robots. The resulting robots are inexpensive but capable. Our experiments have shown that a $50 robot body powered by a smartphone is capable of person following and real-time autonomous navigation. We hope that the presented work will open new opportunities for education and large-scale learning via thousands of low-cost robots deployed around the world.

Smartphones point to many possibilities for robotics that we have not yet exploited. For example, smartphones also provide a microphone, speaker, and screen, which are not commonly found on existing navigation robots. These may enable research and applications at the confluence of human-robot interaction and natural language processing. We also expect the basic ideas presented in this work to extend to other forms of robot embodiment, such as manipulators, aerial vehicles, and watercraft.

A System Overview

Figure 5 depicts the high-level overview of our system. It comprises a smartphone mounted onto a low-cost robot body. The smartphone consumes sensor data (*e.g.* images, IMU, GPS, *etc.*) and optionally user input in order to produce high-level controls for the vehicle such as steering angle and throttle. The microcontroller on the robot body applies the corresponding low-level actuation signals to the vehicle.

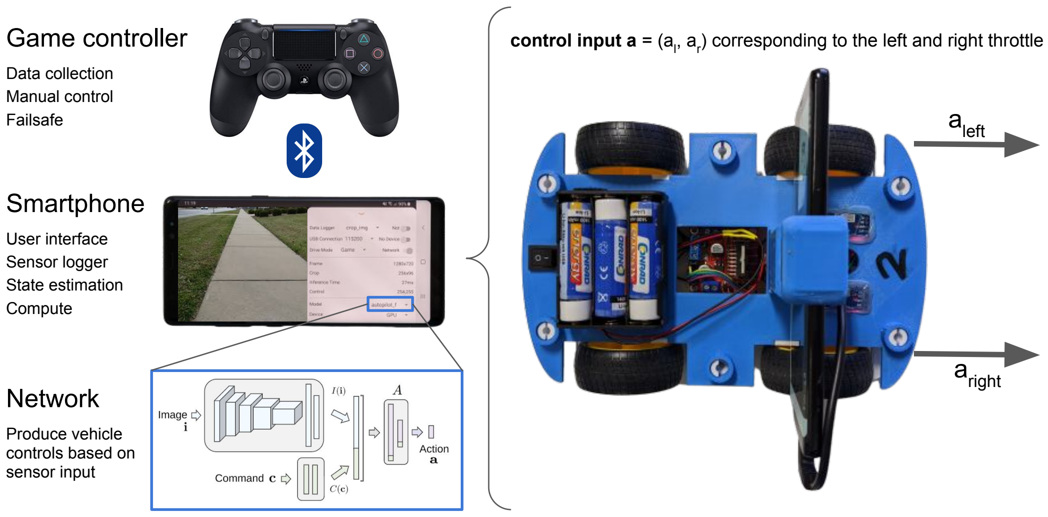

Figure 5: **System overview.** Our wheeled robot leverages a smartphone for sensing and computation. The robot body costs $50 without the smartphone. The platform supports person following and real-time autonomous navigation in unstructured environments.

B Smartphones

Table 7 provides an overview of the smartphones we use in our experiments. We provide the main specifications along with the *Basemark OS II* and *Basemark X* benchmark scores which measure the overall and graphical performance of smartphones. We also include the AI score [21] where available.

Mobile Phone	Release	Price	Main Camera	Memory/RAM	CPU	GPU	Overall	Graphics	AI
Samsung Note 8	09/17	300	12 MP, f/1.7, 1/2.55”	64GB, 6GB	Exynos 8895	Mali-G71 MP20	3374	40890	4555
Huawei P30 Pro	03/19	650	40 MP, f/1.6, 1/1.7”	128GB, 8GB	HiSilicon Kirin 980	Mali-G76 MP10	4654	45889	27112
Google Pixel 4XL	10/19	750	12.2 MP, f/1.7, 1/2.55”	64GB, 6GB	Qualcomm SM8150	Adreno 640	5404	–	32793
Xiaomi Note 8	08/19	170	48 MP, f/1.8, 1/2.0”	64GB, 4GB	Qualcomm SDM665	Adreno 610	2923	17636	7908
Xiaomi Mi 9	02/19	380	48 MP, f/1.8, 1/2.0”	128GB, 6GB	Qualcomm SM8150	Adreno 640	5074	45089	31725
OnePlus 6T	11/18	500	16 MP, f/1.7, 1/2.6”	128GB, 8GB	Qualcomm SDM845	Adreno 630	4941	43886	18500
Samsung Note 10	08/19	750	12 MP, f/1.5-2.4, 1/2.55”	256GB, 8GB	Exynos 9825	Mali-G76 MP12	4544	45007	24924
Huawei P30 Lite	04/19	220	48 MP, f/1.8, 1/2.0”,	128GB, 4GB	Hisilicon Kirin 710	Mali-G51 MP4	2431	20560	-
Xiaomi Poco F1	08/18	290	12 MP, f/1.9, 1/2.55”	64GB, 6GB	Qualcomm SDM845	Adreno 630	4034	43652	6988
Nokia 2.2	06/19	120	13 MP, f/2.2, 1/3.1”	16GB, 2GB	Mediatek MT6761	PowerVR GE8320	848	5669	–

Table 7: **Smartphones.** Specifications of the smartphones used in our experiments. We report the overall, graphics, and AI performance according to standard benchmarks. Top: six smartphones used to collect training data. Bottom: smartphones used to test cross-phone generalization. ”–” indicates that the score is not available.

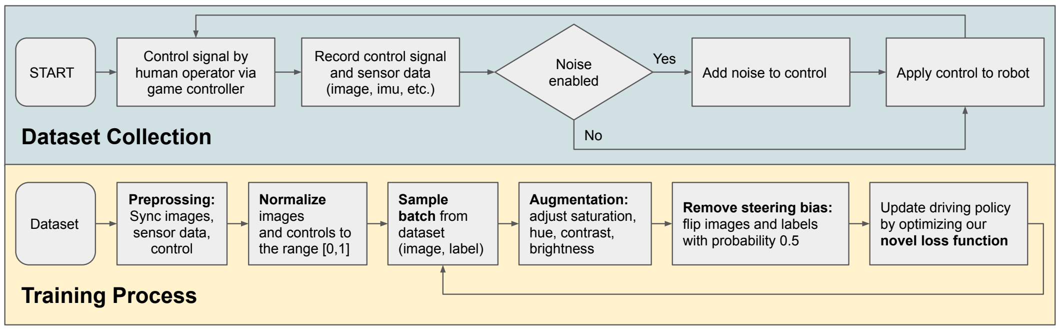
START
Control signal by human operator via game controller
Record control signal and sensor data (image, imu, etc.)
Noise enabled
Yes
No
Add noise to control
Apply control to robot
Dataset Collection
Dataset
Preprossing: Sync images, sensor data, control
Normalize images and controls to the range [0,1]
Sample batch from dataset (image, label)
Augmentation: adjust saturation, hue, contrast, brightness
Remove steering bias: flip images and labels with probability 0.5
Update driving policy by optimizing our novel loss function
Training Process

Figure 6: **Driving policy: Training pipeline.** The flowchart explains the complete process for obtaining our autonomous navigation policy. There are two main components, dataset collection and training the driving policy which is represented by a neural network.

C Autonomous Navigation: Driving Policy Details

Figure 6 depicts the pipeline for obtaining our driving policy. We discuss the details of its different aspects in the following.

C.1 Dataset Collection

We record a driving dataset with a human controlling the robot via a game controller. In previous works, data was often collected with multiple cameras for added exploration [2, 6]. Since we only use one smartphone camera, we inject noise during data collection and record the recovery maneuvers executed by the human operator [6]. We also scatter obstacles in the environment, such as chairs, for added robustness.

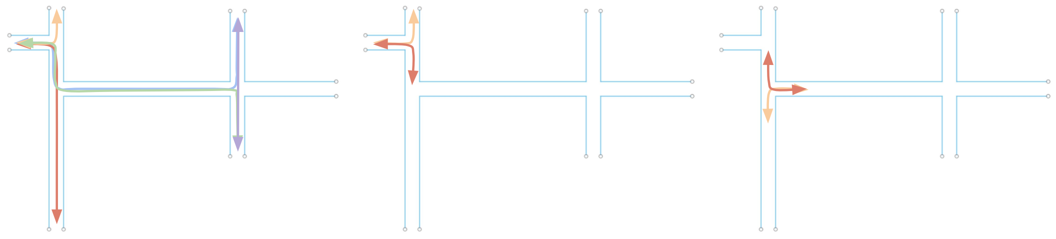

Figure 7: **Training Routes.** We collect data on three different routes: *R1*, *R2* and *R3* (from left to right). *R1* is composed of 5 bi-directional segments with a total of 20 intersections: 8 left turns, 8 right turns, and 4 straights. *R2* and *R3* are two different T-junctions each with two bi-directional segments with a total of two right turns, and two left turns.

We show a map of our training environment in Figure 7 and several images in Figure 8. We define three routes and call them *R1*, *R2* and *R3*. *R1* consists of 5 bi-directional segments with a total of 20 intersections: 8 left turns, 8 right turns, and 4 straights. One data unit corresponds to about 8 minutes or 12,000 frames. *R2* and *R3* both consist of 2 bi-directional segments with a total of two left turns, and two right turns at a T-junction. One data unit corresponds to about 70 seconds or 1,750 frames.

For the experiments in the paper, we collect a dataset with the Xiaomi Mi9 which consists of two data units from *R1* and six data units from both *R2* and *R3*. Half of the data on *R1* is collected with noise and obstacles and the other without. Half of the data on *R2* and *R3* is collected with noise and the other without. The complete dataset contains approximately 45,000 frames corresponding to 30 minutes worth of data.

Figure 8: **Training Environment.** The images depict the environment where the training data was collected.

C.2 Network

Our network is visualized in Figure 9. It takes an image **i** and a command **c** as inputs and processes them via an image module $I(\mathbf{i})$ and a command module $C(\mathbf{c})$. The image module consists of five convolutional layers with 32, 64, 96, 128 and 256 filters, each with a stride of 2, a kernel size of 5 for the first layer, and 3 for the remaining layers. We apply *relu* activation functions, *batch-normalization*, and 20% *dropout* after each convolutional layer. The output is flattened and processed by two fully-connected layers with 128 and 64 units. The command module is implemented as an MLP with 16 hidden units and 16 output units. The outputs of the image module and the command module are concatenated and fed into the control module A which is also implemented as an MLP. It has two hidden layers with 64 and 16 units and then linearly regresses to the action vector a. We concatenate the command **c** with the hidden units for added robustness. We apply 50% *dropout* after all fully-connected layers.

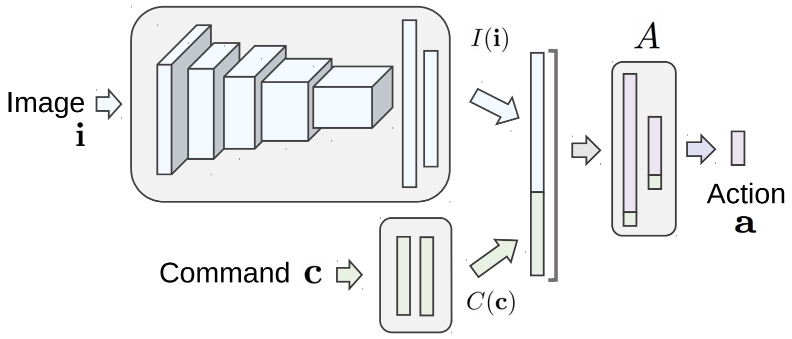

Figure 9: **Driving policy: Network architecture.** Our compact neural network for autonomous navigation runs in real time on most smartphones we tested.

We use an image input size of 256x96, resulting in 1.3M parameters. At the same input resolution, PilotNet [2] has 9.6M parameters and CIL [6] has 10.7M parameters. Our network runs in real time on most smartphones we tested. The average inference times on the Samsung Galaxy Note 10, Xiaomi Mi9, Xiaomi Pocofone F1, and Huawei P30 Lite are 19ms, 21ms, 29ms, and 32ms, respectively. Further speedups are possible by quantization of the network weights and by leveraging the GPU or the recent neural network API (NNAPI) [18].

C.3 Loss function

When training end-to-end driving policies on autonomous navigation datasets, one common challenge is the huge label imbalance. The majority of the time, the vehicle is driving in a straight line, resulting in many images with the same label. One common approach is to resample the dataset or carefully craft individual batches during training [6]. However, this usually requires a fixed dataset or computational overhead. If the dataset is dynamically changing or arrives as a continuous stream these methods are not feasible. Instead, we address this imbalance with a weighted loss. The intuition is simple: the stronger the steering angle, the more critical the maneuver. Hence we use a loss with a weighted term proportional to the steering angle combined with a standard MSE loss on the entire action vector to ensure that throttle is learned as well:

$$\mathbb{L} = w^2 \cdot \text{MSE}\left(s^t, s^p\right) + \text{MSE}\left(\mathbf{a^t}, \mathbf{a^p}\right), \tag{1}$$

where $\mathbf{a^t}$ is the target action, $\mathbf{a^p}$ is the predicted action, s^t is the target steering angle, and $w = (s^t + b)$ with a bias b to control the weight of samples with zero steering angle. Since our vehicle uses differential steering, the action vector consists of a two-dimensional control signal $\mathbf{a} = (a_l, a_r)$, corresponding to throttle for the left and right wheels. We compute the steering angle as $s = a_l - a_r$.

C.4 Training Details

We use the Adam optimizer with an initial learning rate of 0.0003 and train all models for 100 epochs. We augment the images by randomly adjusting hue, saturation, brightness and contrast during training. In addition, we flip images and labels to increase our effective training set size and balance potential steering biases. We normalize images and actions to the range $[0, 1]$.

C.5 Validation Metrics

Another challenge in training autonomous driving policies and evaluating them based on the training or validation loss is the lack of correlation to the final performance of the driving policy [5]. Different action sequences can lead to the same state. The validation loss measures the similarity between target and prediction which is too strict. Hence, we define two validation metrics which are less strict and reduce the gap between offline and online evaluation. The first metric measures whether the steering angle is within a given threshold, which we set to 0.1. The second metric is even more relaxed and only considers whether the steering direction of the target and the prediction align. We find empirically that these metrics are more reliable as the validation loss. However, the correlation to the final driving performance is still weak. We pick the best checkpoint based on the average of these two metrics on a validation set.

C.6 Evaluation Details

We design an evaluation setup that is simple to set up in various environments in order to encourage benchmarking using OpenBot. The only thing needed is a T-junction as shown in Figure 10. We define one trial as six segments comprising two straights, two right turns, and two left turns. We distinguish between closed and open turns, the latter being more difficult. To ensure a simple yet comprehensive comparison, we adopt the following metrics: success, distance, number of collisions and inference speed. Success is a binary value indicating weather or not a segment was completed. The distance is measured along the boundary of a segment without counting the intersections. This way, every segment has a length of 10m and the metric is invariant to different corridor widths. If an intersection is missed, we measure the distance until the beginning of the intersection (*i.e.* 5m). The number of collisions is recorded per segment. We measure the inference time of the driving policy per frame to compute the average FPS. All measurements are averaged across the six segments to obtain the results for one trial. We report the mean and standard deviation across three trials for all metrics. All results in the paper are obtained using this evaluation route.

D Additional Experiments

For the following experiments, we collect multiple data units along route *R1* in the training environment (Figure 7) with multiple robots and smartphones. We consider a total of four datasets; each dataset consists of 12 data units or approximately 96 minutes of data, half of which is collected with noise and obstacles. Two datasets are used to investigate the impact of using different phones and the other two to investigate the impact of using different bodies.

Since these policies are trained on more data, we design a more difficult evaluation route as shown in Figure 11. The route contains the same type of maneuvers, but across two different intersections

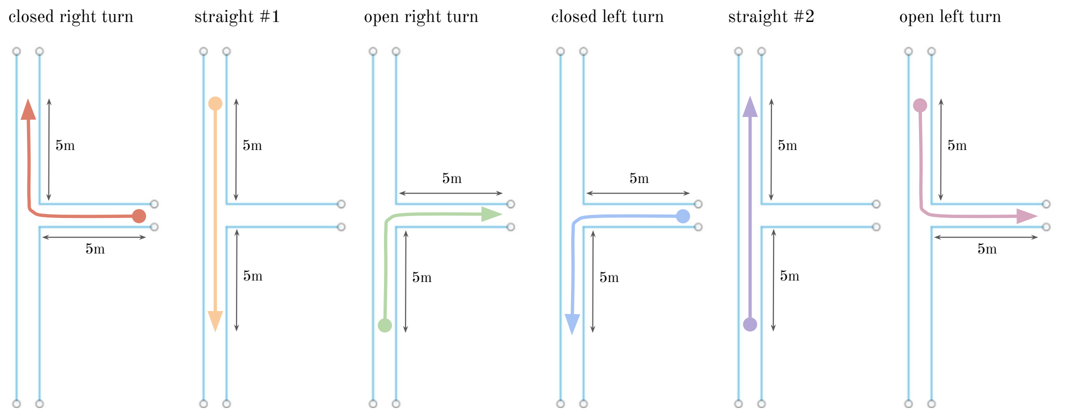

Figure 10: **Evaluation Route 1: T-junction.** Our evaluation route consists of six segments with a total of two straights, two right turns, and two left turns. We report mean and standard deviation across three trials.

and divided into less segments. As a result, small errors are more likely to accumulate, leading to unsuccessful segments and a lower average success rate.

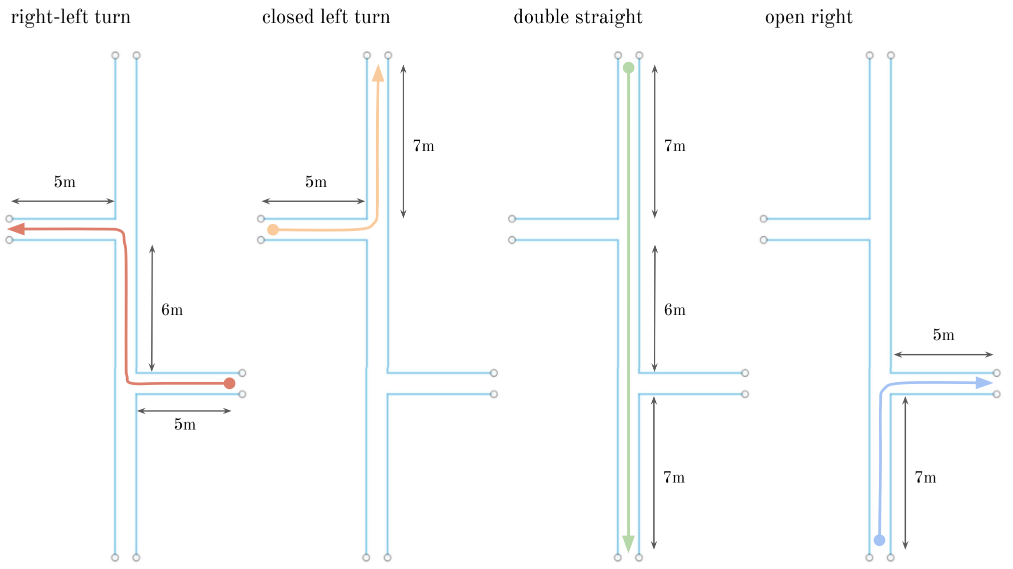

Figure 11: **Evaluation Route 2: Double T-junction.** Our evaluation route consists of four segments with a total of two straights, two right turns, and two left turns across two intersections. We report mean and standard deviation across three trials.

D.1 Learning from data collected with multiple robots

Learning from data collected with multiple smartphones. We investigate whether training on data from multiple phones helps generalization and robustness. We train two identical driving policies, one on data acquired with six different phones (Table 7, top) and another with the same amount of data from only one phone, the Xiaomi Mi9; we keep the robot body the same for this set of experiments. We evaluate both policies on the common training phone, the Mi9. We also evaluate both driving policies on three held-out test phones that were not used for data collection and differ in terms of camera sensor and manufacturer (Table 7, bottom). The P30 Lite has the same camera sensor as the Mi9, but is from a different manufacturer. The Pocofone F1 has a different camera sensor, but is from the same manufacturer. The Galaxy Note 10 differs in both aspects, manufacturer and camera sensor.

Evaluation	Mi9			P30 Lite			Pocofone F1			Galaxy Note 10		
Training	All	Mi9	Δ	All	Mi9	Δ	All	Mi9	Δ	All	Mi9	Δ
Distance (%) ↑	97±5	94±5	**3**	85±19	80±6	**5**	79±7	73±1	**6**	87±11	69±7	**18**
Success (%) ↑	92±14	83±14	**9**	75±25	50±0	**25**	42±14	42±14	**0**	67±14	42±14	**25**
Collisions ↓	0.0±0.0	0.0±0.0	**0.0**	1.0±1.0	0.0±0.0	**1.0**	0.3±0.6	1.3±0.6	**-1.0**	1.7±0.6	1.3±0.6	**0.4**

Table 8: **Autonomous navigation: Transfer across smartphones.** We report the mean and standard deviation across three trials. Each trial consists of several segments with a total of 2 straights, 2 left turns, and 2 right turns.

The results are summarized in Table 8. We find that the driving policy trained on data from multiple phones consistently outperforms the driving policy trained on data from a single phone. This effect becomes more noticeable when deploying the policy on phones from different manufacturers and with different camera sensors. However, driving behaviour is sometimes more abrupt which is reflected by the higher number of collisions. This is probably due to the different field-of-views and positions of the camera sensors making learning more difficult. We expect that this will be overcome with more training data.

We also performed some experiments using the low-end Nokia 2.2 phone, which costs about $100. It is able to run our autonomous navigation network at 10 frames per second. Qualitatively, the driving performance is similar to the other phones we evaluated. However, since it was unable to make predictions in real time, we did not include it in our main experiments, which were focused on the impact of camera sensor and manufacturer.

D.2 Learning from data collected with multiple robot bodies

We also investigate whether training on data from multiple robot bodies helps generalization and robustness. One policy is trained on data collected with three different bodies and another with the same amount of data from a single body; we keep the smartphone fixed for this set of experiments. We evaluate both policies on the common training body, B1, which was used during data collection. We also evaluate on a held-out test body, B4.

The results are summarized in Table 9. We find that the driving policy that was trained on multiple robot bodies performs better, especially in terms of success rate, where small mistakes can lead to failure. The policy that was trained on a single body sways from side to side and even collides with the environment when deployed on the test body. The actuation of the bodies is noisy due to the cheap components. Every body responds slightly differently to the control signals. Most bodies have a bias to veer to the left or to the right due to imprecision in the assembly or the low-level controls. The policy trained on multiple bodies learns to be robust to these factors of variability, exhibiting stable learned behavior both on the training bodies and on the held-out test body.

Evaluation	Body 1			Body 4		
Training	B1-B3	B1	Δ	B1-B3	B1	Δ
Distance (%) ↑	97±5	94±5	**3**	94±5	92±8	**2**
Success (%) ↑	92±14	83±14	**9**	83±14	75±25	**8**
Collisions ↓	0.0±0.0	0.0±0.0	**0.0**	0.0±0.0	0.7±0.6	**-0.7**

Table 9: **Autonomous navigation: Transfer across robot bodies.** We report the mean and standard deviation across three trials. Each trial consists of several segments with a total of 2 straights, 2 left turns, and 2 right turns.

Despite the learned robustness, the control policy is still somewhat vehicle-specific, *e.g.* the differential drive setup and general actuation model of the motors. An alternative would be predicting a desired trajectory instead and using a low-level controller to produce vehicle-specific actions. This can further ease the learning process and lead to more general driving policies.

References

[1] B. Balaji, S. Mallya, S. Genc, S. Gupta, L. Dirac, V. Khare, G. Roy, T. Sun, Y. Tao, B. Townsend, et al. Deepracer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning. *arXiv:1911.01562*, 2019.

[2] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv:1604.07316*, 2016.

[3] M. Bonani, V. Longchamp, S. Magnenat, P. Rétornaz, D. Burnier, G. Roulet, F. Vaussard, H. Bleuler, and F. Mondada. The marxbot, a miniature mobile robot opening new perspectives for the collective-robotic research. In *IROS*, 2010.

[4] Y. Cao, Z. Xu, F. Li, W. Zhong, K. Huo, and K. Ramani. V. ra: An in-situ visual authoring system for robot-iot task planning with augmented reality. In *DIS*, 2019.

[5] F. Codevilla, A. M. Lopez, V. Koltun, and A. Dosovitskiy. On offline evaluation of vision-based driving models. In *ECCV*, 2018.

[6] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy. End-to-end driving via conditional imitation learning. In *ICRA*, 2018.

[7] M. Dekan, F. Duchoň, L. Jurišica, A. Vitko, and A. Babinec. irobot create used in education. *Journal of Mechanics Engineering and Automation*, 3(4):197–202, 2013.

[8] C. Delaunay. Botiful, social telepresence robot for android. https://www.kickstarter.com/projects/1452620607/botiful-telepresence-robot-for-android, 2012. Accessed: 2020-06-20.

[9] DJI Robomaster S1. https://www.dji.com/robomaster-s1. Accessed: 2020-06-20.

[10] GCtronic. Wheelphone. http://www.wheelphone.com, 2013. Accessed: 2020-06-20.

[11] D. V. Gealy, S. McKinley, B. Yi, P. Wu, P. R. Downey, G. Balke, A. Zhao, M. Guo, R. Thomasson, A. Sinclair, et al. Quasi-direct drive for low-cost compliant robotic manipulation. In *ICRA*, 2019.

[12] B. Goldfain, P. Drews, C. You, M. Barulic, O. Velev, P. Tsiotras, and J. M. Rehg. Autorally: An open platform for aggressive autonomous driving. *IEEE Control Systems Magazine*, 39: 26–55, 2019.

[13] J. Gonzales, F. Zhang, K. Li, and F. Borrelli. Autonomous drifting with onboard sensors. In *Advanced Vehicle Control*, 2016.

[14] F. Grimminger, A. Meduri, M. Khadiv, J. Viereck, M. Wüthrich, M. Naveau, V. Berenz, S. Heim, F. Widmaier, J. Fiene, et al. An open torque-controlled modular robot architecture for legged locomotion research. *arXiv:1910.00093*, 2019.

[15] A. Gupta, A. Murali, D. P. Gandhi, and L. Pinto. Robot learning in homes: Improving generalization and reducing dataset bias. In *NeurIPS*, 2018.

[16] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, et al. Searching for mobilenetv3. In *ICCV*, 2019.

[17] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017.

[18] A. Ignatov, R. Timofte, A. Kulik, S. Yang, K. Wang, F. Baum, M. Wu, L. Xu, and L. Van Gool. AI benchmark: All about deep learning on smartphones in 2019. In *ICCV Workshops*, 2019.

[19] S. Karaman, A. Anders, M. Boulet, J. Connor, K. Gregson, W. Guerra, O. Guldner, M. Mohamoud, B. Plancher, R. Shin, et al. Project-based, collaborative, algorithmic robotics for high school students: Programming self-driving race cars at mit. In *ISEC*, 2017.

[20] N. Kau, A. Schultz, N. Ferrante, and P. Slade. Stanford doggo: An open-source, quasi-direct-drive quadruped. In *ICRA*, 2019.

[21] J. Lee, N. Chirkov, E. Ignasheva, Y. Pisarchyk, M. Shieh, F. Riccardi, R. Sarokin, A. Kulik, and M. Grundmann. On-device neural net inference with mobile GPUs. *arXiv:1907.01989*, 2019.

[22] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[23] J. McLurkin, A. McMullen, N. Robbins, G. Habibi, A. Becker, A. Chou, H. Li, M. John, N. Okeke, J. Rykowski, et al. A robot system design for low-cost multi-robot manipulation. In *IROS*, 2014.

[24] Nvidia JetBot. `https://github.com/nvidia-ai-iot/jetbot`. Accessed: 2020-06-20.

[25] M. O'Kelly, V. Sukhil, H. Abbas, J. Harkins, C. Kao, Y. V. Pant, R. Mangharam, D. Agarwal, M. Behl, P. Burgio, et al. F1/10: An open-source autonomous cyber-physical platform. *arXiv:1901.08567*, 2019.

[26] N. Oros and J. L. Krichmar. Smartphone based robotics: Powerful, flexible and inexpensive robots for hobbyists, educators, students and researchers. *IEEE Robotics & Automation Magazine*, 1:3, 2013.

[27] S. Owais. Turn your phone into a robot. `https://www.instructables.com/id/Turn-Your-Phone-into-a-Robot/`, 2015. Accessed: 2020-06-20.

[28] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. F. Chen, C. Choi, J. Dusek, Y. Fang, et al. Duckietown: an open, inexpensive and flexible platform for autonomy education and research. In *ICRA*, 2017.

[29] F. Riedo, M. Chevalier, S. Magnenat, and F. Mondada. Thymio ii, a robot that grows wiser with children. In *IEEE Workshop on Advanced Robotics and its Social Impacts*, 2013.

[30] Romotive. Romo - the smartphone robot for everyone. `https://www.kickstarter.com/projects/peterseid/romo-the-smartphone-robot-for-everyone`, 2012. Accessed: 2020-06-20.

[31] W. Roscoe. An opensource diy self driving platform for small scale cars. `https://www.donkeycar.com`. Accessed: 2020-06-20.

[32] M. Rovai. Hacking a rc car to control it using an android device. `https://www.hackster.io/mjrobot/hacking-a-rc-car-to-control-it-using-an-android-device-7d5b9a`, 2016. Accessed: 2020-06-20.

[33] M. Rubenstein, C. Ahler, and R. Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. In *ICRA*, 2012.

[34] M. Rubenstein, B. Cimino, R. Nagpal, and J. Werfel. Aerobot: An affordable one-robot-per-student system for early robotics education. In *ICRA*, 2015.

[35] A. Setapen. *Creating robotic characters for long-term interaction*. PhD thesis, MIT, 2012.

[36] S. S. Srinivasa, P. Lancaster, J. Michalove, M. Schmittle, C. S. M. Rockett, J. R. Smith, S. Choudhury, C. Mavrogiannis, and F. Sadeghi. Mushr: A low-cost, open-source robotic racecar for education and research. *arXiv:1908.08031*, 2019.

[37] Tensorflow Object Detection Android Application. `https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/android`. Accessed: 2020-06-20.

[38] S. Wilson, R. Gameros, M. Sheely, M. Lin, K. Dover, R. Gevorkyan, M. Haberland, A. Bertozzi, and S. Berman. Pheeno, a versatile swarm robotic research and education platform. *IEEE Robotics and Automation Letters*, 1(2):884–891, 2016.

[39] xCraft. Phonedrone ethos - a whole new dimension for your smartphone. `https://www.kickstarter.com/projects/137596013/phonedrone-ethos-a-whole-new-dimension-for-your-sm`, 2015. Accessed: 2020-06-20.

[40] B. Yang, J. Zhang, V. Pong, S. Levine, and D. Jayaraman. Replab: A reproducible low-cost arm benchmark platform for robotic learning. *arXiv:1905.07447*, 2019.

[41] J. Yim, S. Chun, K. Jung, and C. D. Shaw. Development of communication model for social robots based on mobile service. In *International Conference on Social Computing*, 2010.