Aerial Informatics and Robotics Platform

Shital Shah, Debadeepta Dey, Chris Lovett and Ashish Kapoor
Microsoft Research
One Microsoft Way, Redmond WA 98052 USA

February 15, 2017

Abstract

Machine Learning technologies are increasingly becoming an important tool in building autonomous systems. Learning-based techniques have been useful but typically need a large amount of training data, which is an expensive and time consuming process. Often collecting such data is non-trivial and introduces safety concerns. Consequently, it is becoming increasingly important to be able to accurately simulate the physical environment that autonomous vehicles/robots would operate in. We present a new, easy-to-use simulator that aims to enable designers and developers of a robotic system to seamlessly generate lots of training data. The biggest advantage of this simulator is that it uses recent advances in computation and graphics to simulate the physics and perception such that the environment realistically reflects the actual world. Such realism can enable efficient training and testing of machine learned models by generating vast quantity of ground truth data. One of the key aspects of the fast physics engine is that it enables high frequency simulations with support for hardware-in-the-loop (HIL) as well as software-in-the-loop (SIL) with widely supported protocols (e.g. MavLink). Our cross-platform (Linux and Windows), open-source architecture focuses on being easily extensible to accommodate diverse new types of autonomous vehicles, hardware platforms and software protocols. We use quadrotors as our first autonomous vehicle showcase.

Figure 1: A snapshot from Aerial Informatics and Robotics Platform shows an urban environment in which a UAV is flying. The depth image stream, the materials property view stream and the front camera image stream are shown in real time in the inset.

1 Introduction

The field of robotics is going through a transformation and in recent times methods based on machine learning (ML) and artificial intelligence (AI) have been critical in making rapid progress. For instance, data-driven technologies are some of the key components in autonomous vehicles [1, 2, 3, 4], sociable robots [5, 6] etc. More recently, paradigms such as reinforcement learning [7], learning-by-demonstration [8, 9] and transfer learning [10] are proving a natural means to train various robotic systems. One of the key challenges with these techniques is the high sample complexity - the amount of training data needed to learn useful behaviors is prohibitively high. Moreover, this issue is further exacerbated in the context of recent revolution in deep learning, which often require a very large amount of training data.

Such limitations have significant adverse consequences on robotic systems. One direct consequence is the time a robot needs to learn useful and safe behaviors. However, the bigger issue is that the robot is often non-operational

during the training phase. For example, an autonomous car that is still undergoing training for collision-avoidance, cannot be operated safely. While paradigms such as reinforcement learning promise to alleviate some of these problems by immersing the robot in perpetual learning, it is a challenge to guarantee the safety of actions taken during the training phase.

Simulation remains an integral tool in robotics [11, 12, 13] and promises to provide a valuable means for generating lots of data. One of the key benefits of simulation is the cost and time savings - we can not only create various scenarios, but enact actions at a rapid rate (for example, hundreds of seconds of real-world can be simulated in one second). Secondly, it enables carrying out and studying complex missions that might be time consuming and risky in real-world. Finally, bugs and mistakes in simulation cost virtually nothing - we can crash a vehicle multiple times and thereby get a better understanding of implemented methods under various conditions.

However, simulating the real-world is a big challenge. In order to seamlessly operate in the real-world the robot needs to transfer the learning it does in simulation. Currently, this is a non-trivial task as the difference between simulation and real-world is significant. Simulating the physics of the environment with enough detail is one of the most difficult components [11]. Physics has a profound impact on a host of phenomena including perception, sensing and the dynamics of the system, ground, atmosphere. For example, for robots that aim to use computer vision, it is fairly critical to model the effects of lighting and correctly render reflections, time-of-day and weather-based effects [14]. Similarly, it is critical to develop accurate models of the robotic system dynamics so that simulated behavior closely mimics the real-world.

Aerial Informatics and Robotics Platform is an open-source platform whose aim is to enable such simulation to real-world transfer via rich physics-based models for creating realistic dynamic models as well as rendering the scene. This project is motivated and inspired by several previous simulators (see related work), and one of the key goals is to build a community to push the state-of-the-art.

The platform enables hi-fidelity simulation of a host of physical phenomena such as gravity, magnetism, atmospheric conditions and provide sensor models that attempt to mimic real-life. In addition, this platform exploits recent advances in graphics to build photo-realistic environments, which in turn show promise in training perception modules [15, 16]. One of the key advantages of this framework is that it provides a unified way to do closed-

loop control and planning across a variety of real-world robots. The modular design of the architecture enables definition of a wide variety of vehicles. Currently, the platform contains implementation of aerial robots - more specifically the quadrotor platform.

2 Related Work

Simulators continue to play critical roles in advancing algorithms and ideas in robotics. While a complete review of all the past attempts at simulation is beyond the scope of this paper, we mention a few notable recent works that are closest to our setting.

Gazebo [17] has been one the most popular simulation platforms. It has a wide range of capabilities including a physics engine, a host of sensor models and ability to create 3D virtual worlds. Gazebo goes beyond aerial robots and can be used to build various robots that include manipulator arms, ground vehicles etc. While Gazebo is fairly feature rich it has limitations in terms of photorealism and a legacy physics engine.

Other notable effort includes Hector [18] that combines the dynamic simulation capability of Gazebo with Robotic Operating System [19]. The underlying philosophy is similar to ours, where robotic primitives available via ROS are enabled in the simulation environment provided by Gazebo. Hector only focuses on quadrotor UAVs and has detailed dynamical models. It also has the same limitations in achieving simulation reality as that of Gazebo. Similarly, RotorS [20] provides a modular framework to design Micro Aerial Vehicles, and build algorithms for control and state estimation that can be deployed in the field. RotorS also uses Gazebo as a simulation platform, consequently limiting its reach mostly to low level control loops. Finally, there are many open-source efforts, such as jMavSim [21] that provide hardware-in-the-loop simulation albeit with simple and lightweight multi-rotor models.

3 Architecture

Simulating the real-world is a non-trivial task. Not only are there multiple phenomena that need to be modeled, their complex interactions also needs to simulated realistically. Our design philosophy is depicted in figure 2, where we show the different components and their interactions. In partic-

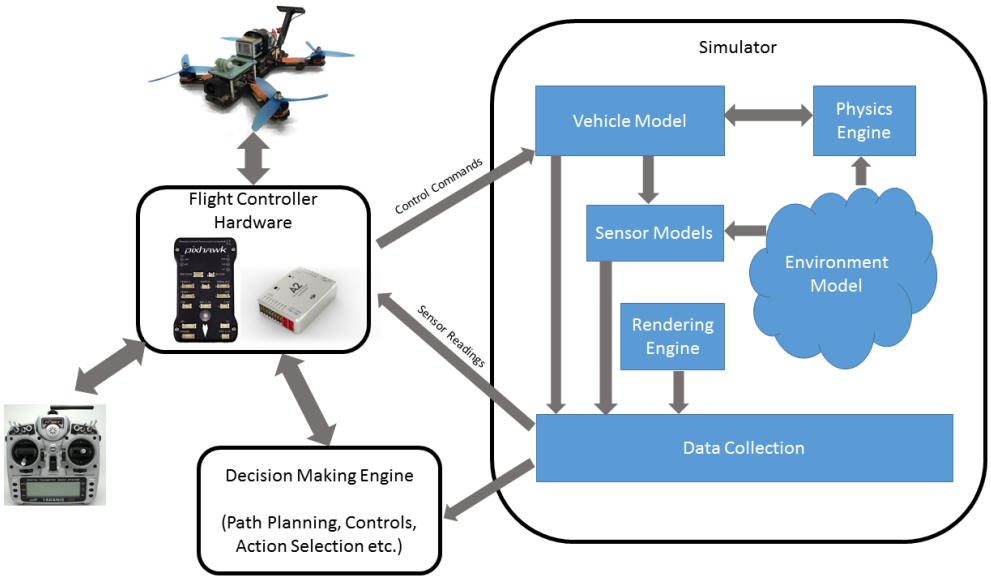

Figure 2: The conceptual architecture of the system that depicts all the components and their interactions. The Aerial Informatics and Robotics Platform allows us to seamlessly transfer between a real quadrotor and the simulated one via the modular design.

ular, the core of the architecture constitutes the simulation environment. At the highest level the simulator contains model of the vehicle, the environment and a physics engine to compute the resulting motions. In addition, our framework also allows support for recording sensor observations that mimic real-world behaviors (such as sensor drift, position errors). Finally, one of the big components is photo-realistic rendering via the Unreal engine [22] that enables computer vision analysis that is transferable to the real-world. The framework also enables hardware-in-the-loop (HIL) simulation, where a flight controller such as Pixhawk [23], directly interacts with the simulation environment. Moreover, this flight controller can be directly connected to an RC controller, or communicate with a *Decision Making Engine* which consists of mapping, localization and planning modules. At the time of writing our current implementation contains both an asynchronous command-line interface as well as easy-to-use APIs to popular autopilots like PixHawk [23] and DJI A3 [24] as well as reactive obstacle avoidance modules. Finally, our framework also enables seamless interface with a physical robot by simple

switch of the communication channel. Below we provide more details on the components of the simulator. Please see the technical documentation for complete definitions.

3.1 Vehicle

The definition of a vehicle primarily entails the information about its total mass, the shape and a set of points where a control input can be applied. The vehicle definition also contains how each of the control inputs maps into a resulting force and a torque at each of the vertices.

Formally, a vehicle of mass m is defined as a collection of K vertices $\{\mathbf{x}_1, .., \mathbf{x}_k\}$, each of which experience a control input $\{u_1, .., u_2\}$. Here $\mathbf{x}_i$ denotes the position of the vertex with respect to the center of mass, and $u_i \in [0, 1]$ the corresponding control command. The shape information entails the 3×3 moment of inertia matrix $\mathbf{I}$ and the 3-dimensional vector $\mathbf{A}$ describes the cross-section area $[A_x, A_y, A_z]^T$. The final ingredients are the equations that map the control inputs u_i to the force ($F_i = f(u_i)$) and the torque ($\tau_i = g(u_i)$) experienced at that vertex i.

Figure 3 shows how a quadrotor can be depicted as a collection of four vertices. The control input u_i is a mapping to the angular speed of each of the propellers located at the four vertices. We can use standard methods to compute moment of inertia and the cross-section areas. Furthermore, the local forces and the torque on the four vertices can be depicted using the following equations:

$$F_i = C_T \sigma \omega_{max}^2 D^4 u_i \quad \text{and} \quad \tau_i = \frac{1}{2\pi} C_{pow} \sigma \omega_{max}^2 D^5 u_i.$$

Here C_T and C_{pow} are the thrust and the power coefficients respectively and are based on the physical characteristics of the propeller, σ is the air density, D is the propeller's diameter and ω_{max} is the max angular velocity in revolutions per minute. Also note that the platform also allows for movements of these vertices. Such a capability to move the points allows us to model changes in thrust direction, such as in a Vertical Take-Off and Landing (VTOL) vehicle and recent quadrotors that change their configuration in flight.

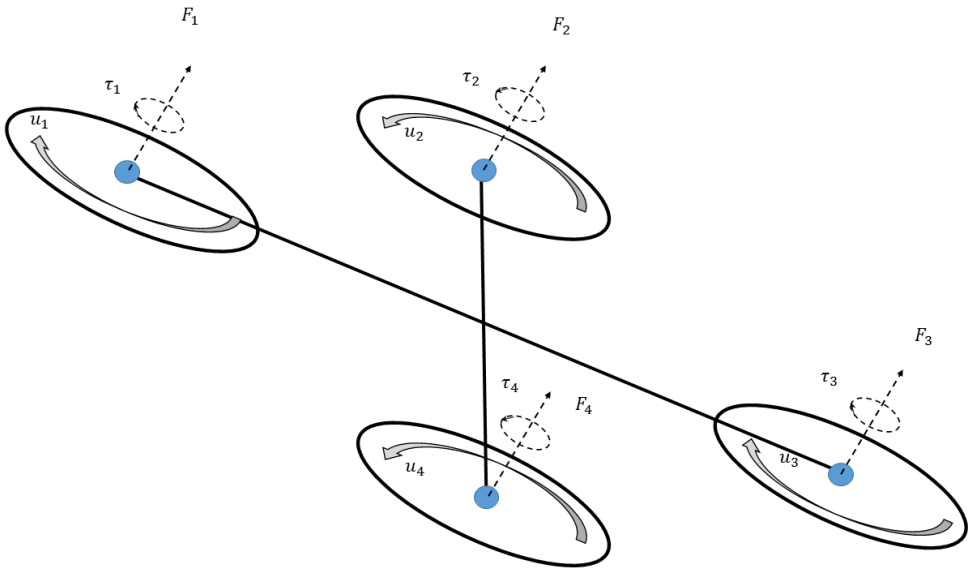

Figure 3: A quadrotor as a vehicle in the framework. The four blue vertices experience the controls $u_1, ..u_4$, which in trun results in the forces $F_1, .., F_4$ and the torques $\tau_1, .., \tau_4$.

3.2 Environment

A vehicle resides and interacts with its environment. There are various physical phenomena such as gravity, air-density, Earth's magnetic field and effects of the air mass and surfaces modeled in the system. The goal here is to provide simulation of these natural phenomena as accurately as possible without using a lot of computation resources. We describe these individual components below.

3.2.1 Gravity

In order to model the effects of gravity accurately, we model the variation of the gravitational acceleration g with altitude. While the variation due to the altitude might be considered negligible in comparison to the radius of the earth, it is important to consider its effect when considering flights when the ground plain itself is at an elevation (e.g. in mountains) or for high altitude flights.

Formally, it is easy to show that the resulting gravitational acceleration

g at height h is related to g_{const} the constant gravitational acceleration on surface of the earth according to:

$$g = g_{\text{const}} \cdot \frac{R_e^2}{(R_e + h)^2} \approx g_{\text{const}} \cdot (1 - 2\frac{h}{R_e}).$$

Here, R_e is the radius of the earth and the first equality arises by simple algebraic manipulation of the Newton's law of gravity. The approximation allows us to circumvent the expensive square operation and is a consequence of applying binomial theorem and neglecting the higher powers.

3.2.2 Magnetic Field

Modeling Earth's magnetic field is a very important but complex task. Magnetometer is one of the key sensors in most of the autonomous vehicles but given the complexities involved it is not surprising that none of the mainstream simulators attempt to accurately model this effect. One of the best publicly available magnetic field computation code is the World Magnetic Model (WMM) model [25] produced by National Oceanic and Atmospheric Administration (NOAA). Unfortunately, this model requires expensive computational resources as well as large memory to load big lookup tables. Consequently, it is non-trivial to use the WMM model in high frequency simulation (as well as lightweight autonomous vehicles). One of the core contributions of this simulator is an efficient component that models Earth's magnetic field accurately.

We start with a tilted dipole model of Earth where we assume Earth as perfect dipole sphere and ignore all but the first order terms of the magnetic field. We can get by with such an approximation because instead of an absolute value, we are interested only in the variation of the magnetic field as we move in space.

Given a geographic latitude θ, longitude ϕ and altitude h (from surface of the earth), we first compute the magnetic co-latitude θ_m using:

$$\cos\theta_m = \cos\theta\cos\theta^0 + \sin\theta\sin\theta^0\cos(\phi - \phi^0).$$

Where θ^0 and ϕ^0 denote the latitude and longitude of the true magnetic north pole. Then, the total magnetic intensity $|B|$ is computed as:

$$|B| = B_0(\frac{R_e}{R_e + h})^3\sqrt{1 + 3\cos^2\theta_m}$$

Here B_0 is the mean value of the magnetic field at the magnetic equator on the Earth's surface, θ_m is the magnetic co-latitude and R_e is the mean radius of the Earth. Next, we determine the inclination α and declination β angles via the following relationship:

$$\tan\alpha = 2\cot\theta_m \quad \text{and} \quad \sin\beta = \begin{cases} \sin(\phi-\phi^0)\frac{\cos\theta^0}{\cos\theta_m}, & \text{if } \cos\theta_m > \sin\theta^0\sin\theta \\ \cos(\phi-\phi^0)\frac{\cos\theta^0}{\cos\theta_m}, & \text{otherwise.} \end{cases}$$

Finally, we can compute the horizontal (H), the vertical (Z), the latitudnal (X) and the longitudnal (Y) components of the magnetic field vector as follows:

$$H = |B|\cos\alpha \qquad Z = |B|\sin\alpha$$
$$X = H\cos\beta \qquad Y = H\sin\beta.$$

3.2.3 Air Pressure and Density

Both air pressure and density have a profound effect on the behavior of the vehicle both due to their effect on the amount of thrust, lift and drag, a moving surface experiences. Especially, it important to model the changes in the air pressure and density as the altitude changes.

In our implementation we use the 1976 U.S. Standard Atmosphere model [26] for altitude below 51 kilometers, which switches to the model proposed in [27] beyond that. Air density at pressure P and temperature T is then calculated as $\sigma = \frac{P}{R \cdot T}$, where R is the specific gas constant.

3.3 Physics Engine

The goal of the physics engine is to consider the vehicle and the environment it operates in and determine the resulting motion due to the forces and torques that come into play as control is applied. While there are many off-the-shelf physics engines that could have been used, the platform implements its own module to simulate physics. This was done primarily to address computational efficiency.

For real-time robotics applications it is important that the physics engine operates at a very high frequency - our empirical assessment showed that we needed to run such an engine at approximately 1000 Hz. Most of the existing engines do not operate at such high frequency. One of the primary

reasons is that in these engines collision checking procedure ends up taking a lot of compute. We alleviate this problem by delegating the task of collision checking to the visual rendering engine (see section 3.5), thereby focusing just on the core aspects of the physics computation.

3.3.1 Linear and Angular Drag

Besides considering the forces F_i and torques τ_i as described earlier in section 3.1, the physics engine also computes the parasitic drag and torque resulting due to the movement of the vehicle in the air. Given, the vehicle cross-section areas (in vehicle description). The linear parasitic drag can be computed as:

$$F_d = \frac{1}{2}\sigma v^2 C_{lin} A.$$

Here C_{lin} is the linear air drag coefficient. Similarly, we compute torque τ_d due to this drag. While computing drag requires detailed description of the vehicle body, in the current implementation we perform torque computations for box-sized objects.

The physics engine, thus, considers all the forces F_i, F_d and all the torques τ_i and τ_d in order to determine the resulting motion for the next time tick. This high-frequency computation allows us to compute realistic dynamic trajectories even for a highly non-linear system.

3.4 Sensors

One of the main goals of this work is to enable machine learning for real-world systems by collecting training data. Thus, it is imperative to also model the sensors collecting the data as realistically as possible. Because the simulator is aware of the ground truths by definition, the sensors can mimic how the robot perceives the environment. Below we describe how the sensors were implemented in order to realistically reflect their individual idiosyncrasies.

3.4.1 Barometer

Most of the current simulators have very simple models for the barometer. Specifically, a majority of the models simply just report Gaussian noise added to the ground truth. There are at least two aspects of the real barometers that need to be modeled. First, in nature the air pressure varies with time,

which introduces a bias in the barometric reading. Secondly, this bias also drifts, consequently the barometer output slowly changes with time.

We model this drifting bias via a Gaussian Markov process [28]. Formally, if we denote the current bias factor as b_k then the drift is modeled as:

$$b_{k+1} = w \cdot b_k + (1 - w) \cdot \eta, \text{where: } w = e^{\frac{-dt}{t}} \text{ and } \eta \sim N(0, s^2).$$

Here t, is the time constant for the process and set to 1 hour in our model. η is a zero mean Gaussian noise with standard deviation $s = 1.8$. This choice is reasonable as at the sea level the mean standard deviation is widely accepted to be 1.8% [29].This drifting bias is incorporated in the barometer by first converting the true altitude h to the corresponding pressure reading p. The biased incorporated pressure reading is computed as $p^{\text{out}} = p(1+b_k)$, which in turn is transformed back to the altitude h^{out} and reported as the barometer's reading.

3.4.2 Gyroscope and Accelerometer

Gyroscope and accelerometers constitute the core of the inertial measurement unit (IMU). Given the central role of IMU in design of autonomous system, we payed a lot of attention to simulate this aspect realistically [30]. Allan Variance (or Deviation) Plots are a common way to describe the IMU noise and we use such plots to model the sensor behavior.

From the Allan Variance plots we get information about the angular random walk (ARW) denoted as r_a, the stability bias b_0 and the time constant t_a. Given the true angular velocity ω, we compute the output angular velocity ω^{out} recorded by the sensor as:

$$\begin{aligned} \omega^{\text{out}} &= \omega + \eta_a + b_t, && \text{where } \eta_a \sim N(0, r_a) \text{ and} \\ b_t &= b_{t-1} + \eta_b, && \text{where } \eta_b \sim N(0, b_0\sqrt{\frac{dt}{t_a}}). \end{aligned}$$

Similarly, from the plots we also get information about the velocity random walk (VRW) and the associated stability bias and the time constant. A transformation analogous to the one mentioned above is used to compute the output velocity v^{out} from the ground truth. The current implementation uses the data-sheet from InvenSense MPU-3300 unit. Alternative IMU's can be supported by replacing the relevant constants in the implementation.

3.4.3 Magnetometer

Given the detailed simulation of magnetic field as described in section 3.2.2, it is straightforward to simulate the magnetometer. Given the geographic coordinates we use the above mentioned dipole model to get the components of the magnetic field. Sensor dependent Gaussian noise and a fixed bias are added to these readings and the output is produced in relation to the body frame of the vehicle. In our implementation we use the data sheet for STM Microelectronics LSM303D magnetometer. The simulation framework enables definition of alternate or newer devices as well.

3.4.4 Global Positioning System (GPS)

The GPS model in the simulator is a straightforward implementation with essential noise components and other sources of errors. One of the key problems with GPS sensors is latency. We implement the typical latency of 200 ms (i.e. position currently indicated by GPS was only true 200 ms ago). Additionally, the GPS sensor has much slower update rate at typically 20-50 ms, when compared to the other sensors. We implement these critical aspects in simulation as often GPS is used to correct drifts in IMU, consequently inaccuracies in GPS model can cascade down to other aspects.

We also simulate position error estimates as computed by the GPS sensor itself. Specifically, our simulation of the GPS module outputs the standard deviation of horizontal and vertical position error estimated by the sensor itself. The main characteristic of these two outputs is that they should decay with time as the GPS fix gets better. We simulate this delay using a simple first order low pass filter. Finally, most GPS sensors cannot reliably measure vertical velocities, consequently we limit the simulation to the velocities in the horizontal plane.

3.5 Visual Rendering

Since photo-realistic rendering was a key requirement for Aerial Informatics and Robotics Platform we chose the popular, recently open-sourced Unreal Engine 4 (UE4) [22] as the rendering pipeline. UE4 brings several key features which made it an attractive choice:

- **Photo-realistic rendering**: UE4 brings some of the most cutting-edge graphics advances to the table including real-time global illumi-

Figure 4: The figure above shows various images from the simulator including the UAV model, urban and forest scenes as imaged by the front-facing camera of the UAV.

nation. Figure 4 shows a few screen-shots from Aerial Informatics and Robotics Platform which highlight the near to real-world quality of the rendering.

- **Open-source and cross-platform**: UE4 is completely open-source, free to use for simulation, and works seamlessly on Linux, Windows, iOS, Android and OS X. It is also very well-documented and comes with a rich set of developer tools thus making the development of Aerial Informatics and Robotics Platform on top much easier.

- **Large environment marketplace**: The large marketplace [31] where one can buy pre-made elaborate environments, make it easy to collect data from a variety of environments.

3.5.1 Monocular and Stereo Camera Simulation

In Aerial Informatics and Robotics Platform we have added functionality to add virtual monocular and stereo cameras to any vehicle and receive the RGB and depth image streams in the framework of choice (e.g. ROS). See Figure 1 for an example stereo image pair and corresponding depth image. This functionality is crucial for collecting large amounts of perception data in realistic environments.

4 Conclusion

The goal of the Aerial Informatics and Robotics Platform is to enable rapid training and development of data-driven robotic systems. In particular, the platform enables hi-fidelity simulation which in turn can be used to collect training data for building machine learning models. The core components include a fast physics engine with detailed models of physical phenomenon, and a photo-realistic perception engine that enables training and testing of computer vision modules. By leveraging such a simulator, we hope to effectively utilize methods such as reinforcement learning and imitation learning and enable simulator to real-world transfer of machine learned technologies.

References

[1] Chris Urmson, Charlie Ragusa, David Ray, Joshua Anhalt, Daniel Bartz, Tugrul Galatali, Alexander Gutierrez, Josh Johnston, Sam Harbaugh, William Messner, et al. A robust approach to high-speed navigation for unrehearsed desert terrain. *Journal of Field Robotics*, 23(8):467–508, 2006.

[2] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[3] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692, 2006.

[4] J.A. Bagnell, D. Bradley, D. Silver, B. Sofman, and A. Stentz. Learning for autonomous navigation. *Robotics Automation Magazine, IEEE*, June 2010.

[5] Hideki Kozima, Marek P Michalowski, and Cocoro Nakagawa. Keepon. *International Journal of Social Robotics*, 1(1):3–18, 2009.

[6] Stephanie Rosenthal, Joydeep Biswas, and Manuela Veloso. An effective personal mobile robot agent through symbiotic human-robot in-

teraction. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 915–922. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[7] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[8] Yang Gao, Jan Peters, Antonios Tsourdos, Shao Zhifei, and Er Meng Joo. A survey of inverse reinforcement learning techniques. *International Journal of Intelligent Computing and Cybernetics*, 5(3):293–311, 2012.

[9] J. Andrew (Drew) Bagnell. An invitation to imitation. Technical Report CMU-RI-TR-15-08, Robotics Institute, Pittsburgh, PA, March 2015.

[10] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[11] Abhijeet Tallavajhula and Alonzo Kelly . Construction and validation of a high fidelity simulator for a planar range sensor. In *IEEE Conference on Robotics and Automation*, May 2015.

[12] Ioan A Sucan and Sachin Chitta. Moveit! *Online at http://moveit. ros. org*, 2013.

[13] Rosen Diankov and James Kuffner. Openrave: A planning architecture for autonomous robotics. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, 79, 2008.

[14] Srinivasa G Narasimhan and Shree K Nayar. Shedding light on the weather. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2003.

[15] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *European Conference on Computer Vision*, pages 102–118. Springer, 2016.

[16] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2016.

[17] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE, 2004.

[18] Johannes Meyer, Alexander Sendobry, Stefan Kohlbrecher, Uwe Klingauf, and Oskar Von Stryk. Comprehensive simulation of quadrotor uavs using ros and gazebo. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 400–411. Springer, 2012.

[19] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, 2009.

[20] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. Rotorsa modular gazebo mav simulator framework. In *Robot Operating System (ROS)*, pages 595–625. Springer, 2016.

[21] Jmavsim. `https://pixhawk.org/dev/hil/jmavsim`.

[22] Brian Karis and Epic Games. Real shading in unreal engine 4. In *Proc. Physically Based Shading Theory Practice*, 2013.

[23] Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys. Pixhawk: A system for autonomous flight using onboard computer vision. In *Robotics and automation (ICRA), 2011 IEEE international conference on*, pages 2992–2997. IEEE, 2011.

[24] Dji a3 autopilot. `http://store.dji.com/product/a3?from=menu_products`.

[25] A. S. Chulliat, P. Macmillan, C. Alken, M. Beggan, B. Nair, A. Hamilton, V. Woods, S. Ridley, Maus, and A. Thomson. The us/uk world

magnetic model for 2015-2020: Technical report. *NOAA National Geophysical Data Center, Boulder*, 2015.

[26] Roland Stull. *Practical Meteorology: An Algebra-based Survey of Atmospheric Science*. University of British Columbia, 2015.

[27] `http://www.braeunig.us/space/atmmodel.htm`.

[28] A. M. Sabatini and V. Genovese. A stochastic approach to noise modeling for barometric altimeters. *Sensors (Basel, Switzerland)*, 13(11), 2013.

[29] David Burch. *Mariner's Pressure Atlas: Worldwide Mean Sea Level Pressures and Standard Deviations for Weather Analysis and Tropical Storm Forecasting*. Starpath School of Navigation, 2014.

[30] O. J. Woodman. An introduction to inertial navigation. *University of Cambridge*, 2007.

[31] Unreal engine marketplace. `https://www.unrealengine.com/marketplace/content-cat/assets/environments`.