

# Nano-backpack designed for humanoid NAO robot

Barth, Alexander

M.Sc. Student Computer Engineering

Wagner, Royden

M.Sc. Student Computer Engineering

Institute for Computer Engineering (ZITI)

University of Heidelberg

## Abstract

Nano-backpack is an open hardware add-on for the humanoid robot NAO. It adds a Nvidia Jetson Nano development board to the robot and thereby enhances the computational capabilities of the system. The corresponding software architecture includes a webapp to control the robot and allows an easy start into the development of further applications. The source code and CAD models are available on GitHub: <https://github.com/roydenwa/nano-backpack>  
An in-depth instruction manual is provided in an additional document.

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Related work</b>	<b>1</b>
2.1	NAO Backpack . . . . .	1
2.2	Monocular Depth Estimation . . . . .	2
<b>3</b>	<b>Mechanical design</b>	<b>2</b>
<b>4</b>	<b>Remote access and control NAO robot</b>	<b>4</b>
4.1	Architecture . . . . .	4
4.2	Webapp . . . . .	5
<b>5</b>	<b>Depth estimation in video stream</b>	<b>5</b>
<b>6</b>	<b>Conclusion and outlook</b>	<b>8</b>

## List of Figures

1	Encoder-decoder architecture [2]	2
2	Nano-backpack	2
3	Bottom-cover of Nano-backpack and the standard battery-cover	3
4	System overview	4
5	Webapp	5
6	Testing the depth estimator	7
7	Depth-map for same scene with a person in background	7

## List of Source codes

1	Custom dataclass - Video	6
2	Custom decorator - tf_inference	6

## 1 Introduction

This work is based on the humanoid and programmable robot NAO. NAO robots are developed by SoftBank Robotics and have become a standard in today's education and research. The robot's operating system is called NAOqi. SoftBank provides an API with several SDKs to interact with the NAO robot and develop applications via NAOqi.

The Nano-backpack is designed to add a Jetson Nano to the NAO to improve the robot's computing power and to simplify accessing the robot. Jetson Nano is a single board computer developed by NVIDIA for embedded applications focused on machine learning and robotics. The Jetson Nano runs a Linux operating system and has a dedicated GPU, so GPU-versions of modern machine learning frameworks can be used for training and inference of machine learning models.

This work describes the mechanical design of the backpack, a webapp designed to control the NAO robot and provides the basis for a dedicated machine learning use-case for the system.

## 2 Related work

This section provides an overview of related work.

### 2.1 NAO Backpack

Softbank's NAO may be today's standard in education and research and provides advantages in comparison with other available humanoid platforms. Nevertheless it's advantages come along with some drawbacks. The on-board processing power is one of the strongest limitations. Also some of the earlier software works with NAO are outdated. Therefore [1] introduce a open-hardware accesory for the NAO called *NAO Backpack*. With this backpack, the NAO is able to carry more powerful hardware and overcomes the previous limitations of testing with high wlan latency or long ethernet cables. The hardware upgrade creates a wide range

of possibilities in different research areas like computer vision because of the new hardwares' GPUs. Henceforth more computational intensive algorithms can be used with the NAO.

## 2.2 Monocular Depth Estimation

Depth estimation is a computer vision task, where one or multiple input-images of a scene are converted into depth maps of that scene. The output of such a system is usually visualized with pseudo-colored depth images where different colors mark distances within the input-image.

The standard approach for depth estimation is to use two stereo-images captured by a stereo-camera and calculate distances within the scene using triangulation. A recent work showed that monocular depth estimation using machine learning can also lead to good results, although it uses only a single input-image of a scene. [2]

Alhashim and Wonka propose a fairly simple encoder-decoder architecture for this task:

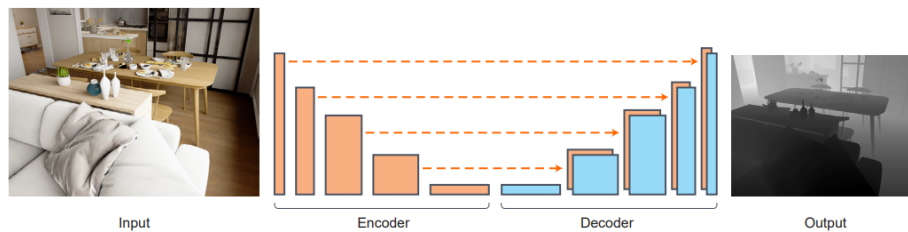
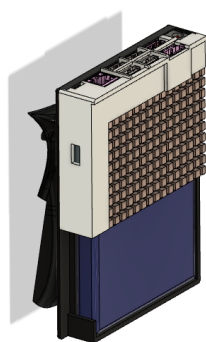


Figure 1: Encoder-decoder architecture [2]

The used encoder is a popular pretrained convolutional neural network, called DenseNet-169. The filtered information is then fed into a decoder structure, consisting of multiple up-sampling layers to reconstruct the pseudo-colored depth image. Additional skip-connections, which transfer information from the first layers directly to the last layers, further improve the performance.

## 3 Mechanical design



(a) CAD model



(b) Test unit mounted on NAO robot

Figure 2: Nano-backpack

The Nano-backpack is designed to carry a Nvidia Jetson Nano development-board and a LiPo-battery to power it. It consists of three major parts, a bottom-cover, a top-cover and a flat copper heat sink, which completes the top-cover.

The bottom-cover is colored black in Figure 2 (a). In comparison to the ODROID development-board used by [1] the Jetson Nano has twice the amount of RAM and a much more powerful dedicated Nvidia GPU on board. This increase in computational capabilities comes at the cost of a bigger footprint and a higher power consumption. Therefore the back-plate of [1], which replaces the standard battery-cover of the NAO robot, is reused in the bottom-cover, but the rest of it is redesigned to fit the bigger components.



Figure 3: Bottom-cover of Nano-backpack and the standard battery-cover

In the upper area of the bottom-cover there is space for the Jetson Nano and holes for its attachment with screws. In the lower area of the bottom-cover there is space for a LiPo battery manufactured by AINOPE. This battery model has a capacity of 5000 mA h and is able to power the Jetson Nano operating in 5 W or in 10 W mode. [Appendix B] The battery is mounted inside the bottom-cover using double sided tape.

The top-cover is colored white in Figure 3 (a) and it is designed to allow access to the various connectors of the Jetson Nano. For easy access, all those connectors are facing upwards, when the backpack is mounted on a NAO robot. In Detail the Nano-backpack is intended to be used with a Ethernet cable connecting the Jetson Nano and the NAO robot and a Wifi dongle for accessing the Jetson Nano via Secure Shell, ssh. The top-cover also has a cut-out on the side in which a switch can be mounted to switch the Jetson Nano on and off.

The top- and bottom-cover in the test unit of the Nano-backpack are 3D-printed with PETG at Heideleberg University.

The top view of the Nano-backpack is completed by a newly designed heat sink for the Jetson Nano. This custom heat sink is made out of pure copper and is significantly lower and wider than the standard heat sink that comes with the Jetson Nano. Overall, the new heatsink is used as cover and is also better suited for use as a passive heat sink due to its material and design.

In Detail, the standard heat sink is made out of AL6061-T6 aluminium, has a volume of  $1.64 \times 10^4 \text{ mm}^3$  and a surface area of  $1.77 \times 10^4 \text{ mm}^2$ , the new designed heat sink has a volume of  $1.4 \times 10^4 \text{ mm}^3$  and a surface area of  $1.94 \times 10^4 \text{ mm}^2$ . Consequently, the surface area is 9.6% larger, which improves heat conduction to the surrounding air. In addition, the

new heat sink is better suited to conduct heat away from the heat source due to its larger footprint and the lower thermal resistance in copper in comparison to aluminium. [3]

Furthermore, the new heat sink for the test unit, like the standard one, is painted black to protect it from corrosion. The newly designed heat sink is mounted on the Jetson Nano with six screws, heat conducting tape and a special thermal transfer plate designed by Connect Tech Inc. The thermal transfer plate is designed to fit the surface of the Jetson Nano Chip. [Appendix C]

## 4 Remote access and control NAO robot

This section describes a webapp and the surrounding architecture on the Nano-backpack.

### 4.1 Architecture

The next figure shows the interaction of hardware and software components that enable the development of complex applications for the NAO robot with the Nano-backpack:

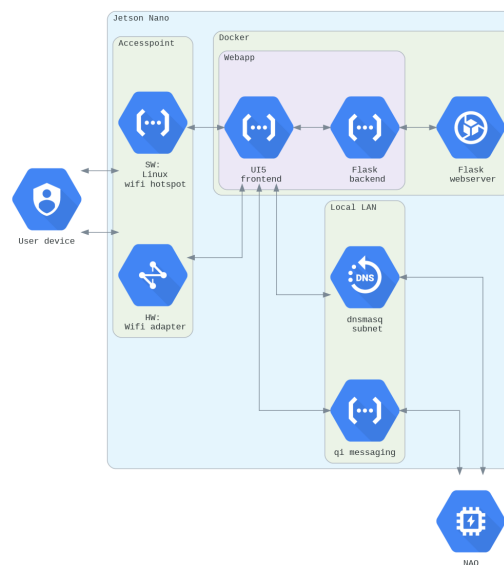


Figure 4: System overview

With a N150 Wifi dongle the Jetson Nano is used as wifi hotspot. Software-wise this hotspot is created with every boot-process using the open source library linux-wifi-hotspot by Lakindu Akash. [4] This creates a local wifi network to which you can connect with a laptop or cell phone, for example. If you are connected to this wifi network, you can access the Jetson Nano via Secure Shell, ssh, view or modify files and even execute programs. If you are connected you can also see webapps running on the Jetson Nano if you navigate to the right URL in a browser on your own device. Thereby a ssh-connection can be used to start or modify the provided webapp or to develop and run own webapps on the Nano.

In Detail, the provided web app runs inside a Docker container on the Jetson Nano. Docker is a software tool which virtualizes operating systems and is very useful and common in web

development. It can be used to encapsulate a webapp, the needed software dependencies and a suitable "virtual" operating system. [5]

The Ethernet connector of the Jetson Nano is used to connect it to the NAO robot via Ethernet cable. Software-wise this connection is managed with the open-source library *dnsmasq* and a custom bash-script. The custom script generates local sub-network for a given IP-address-range and provides a Domain Name System, DNS, for it. Thereby webapps running on the Jetson Nano can exchange messages and data with the NAO robot.

## 4.2 Webapp

The provided webapp has a back-end written in Python with the web-framework Flask and a front-end written in JavaScript with the framework OpenUI5. Flask is an open source web-framework written by Armin Ronacher that can be used to template HTML or to render pre-build templates provided by another framework. OpenUI5 is an open source framework provided by SAP that can be used to create responsive user interfaces for different devices. For the communication of the webapp with the NAO robot the *qi2* SDK for the NAOqi API is used. This SDK is the newest one written in JavaScript.

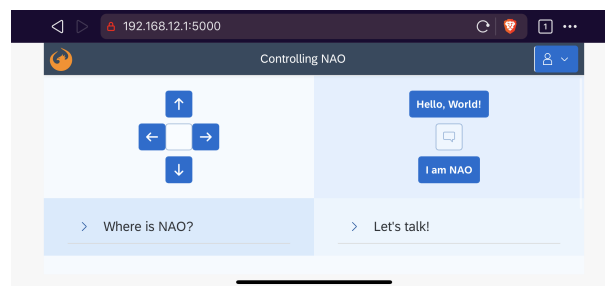


Figure 5: Webapp

The control pad on the left side can be used to control the robot via the *ALMotion* service.

The buttons on the right can be used to let the robot say predefined quotes via the *ALTextToSpeech* service. The two tiles below are placeholders for further functions.

## 5 Depth estimation in video stream

Alhashim and Wonka provide functions to use their pretrained depth estimator on single images.[2] Following a method is described, how video data can be analyzed with the pre-trained depth estimator.

This method is based on the open-source computer vision library OpenCV. OpenCV was originally written in C/C++ but in this method OpenCV-Python is used, which is a wrapper that makes the functionality of OpenCV available in Python.

A custom Python-dataclass is used to handle video-input:

```

1  @dataclass
2  class Video:
3      '''
4      Dataclass to manage video input with cv2
5      src = 0 -> webcam
6      '''
7      src: int
8      width: float
9      height: float
10     cap: object = field(init=False)
11
12     def __post_init__(self):
13         self.cap = cv2.VideoCapture(self.src)
14         self.cap.set(3, self.width)
15         self.cap.set(4, self.height)
16
17     @tf_inference
18     def get_frame(self):
19         ret, frame = self.cap.read()
20         if ret:
21             return frame

```

Listing 1: Custom dataclass - Video

This dataclass wraps the VideoCapture class of OpenCV and adds a custom decorator for the machine learning inference (line 17). Alhashim and Wonka use the machine learning library TensorFlow to implement their depth estimator, following all TensorFlow related variables and functions contain the abbreviation "tf". The decorator `@tf_inference` wraps the `get_frame()` function and thereby applies the machine learning inference to every single frame of an input-video:

```

1  def tf_inference(func, max_depth=1000, min_depth=10):
2      ''' tf decorator func for Video class '''
3      def func_wrapper(*args, **kwargs):
4          frame = func(*args, **kwargs)
5
6          if "model" in globals():
7              # convert frame for tf and depth-model
8              frame_tf = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
9              frame_tf = np.clip(np.asarray(frame_tf, dtype=float) / 255, 0, 1)
10             frame_tf = frame_tf[np.newaxis, ...]
11
12             # get predictions and convert back to img
13             predictions = model.predict(frame_tf, batch_size=2)
14             depth_frame_raw = predictions[0, :, :, :]
15             depth_frame_raw *= 255.0 / depth_frame_raw.max()
16             depth_frame_raw = np.array(depth_frame_raw, dtype=np.uint8)
17
18             # pseudo-colorize:
19             depth_frame_color = cv2.applyColorMap(depth_frame_raw, cv2.COLORMAP_PLASMA)
20
21             return frame, depth_frame_color
22         else:
23             return frame
24     return func_wrapper

```

Listing 2: Custom decorator - tf\_inference

The decorator also checks if a machine learning model is provided as global variable and

returns just the frame if no model is provided. The custom dataclass *Video* can therefore be used with and without machine learning to manage video-input. Finally the pseudo-colored depth-frame is created by applying the `applyColorMap()` function of OpenCV to the predicted depth-frame.

Alhashim and Wonka provide two pretrained versions of their depth estimator, one trained on the KITTI dataset and one trained on the NYU Depth V2 dataset. The KITTI dataset is intended for autonomous driving and therefore features images of street scenes, the NYU Depth V2 features images of indoor scenes. NAO robots are mostly used indoors, so the on NYU Depth V2 dataset trained model fits this use-case.

The next figure shows how this pretrained model performs on unseen data. For pseudo-coloring the OpenCV color-map *COLORMAP\_PLASMA* is used, where distant objects are marked violet and close objects yellow:

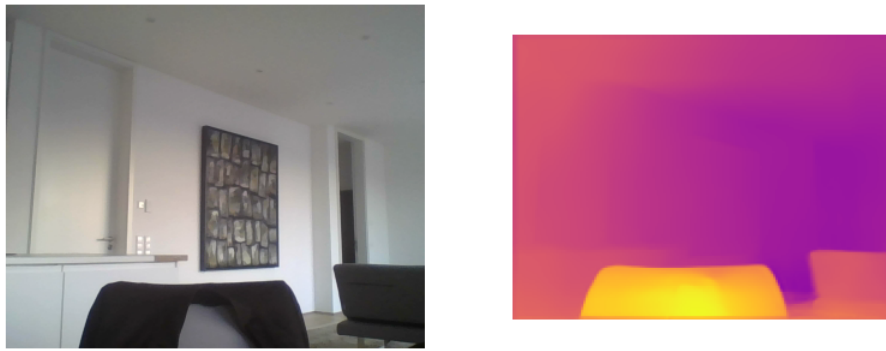


Figure 6: Testing the depth estimator

It detects correctly that the chair and the couch are relatively close to the camera. It also correctly detects that the walls are clearly behind it and the point furthest away is in the middle of the right side. All in all, the level of detail is rather low.

It is interesting that especially humans are well recognized, in the following picture is the same scene with a person in the background:



Figure 7: Depth-map for same scene with a person in background

The NYU dataset contains RGB-images with a width of 640 pixels and a height of 480 pixels. The performance tests show that the pre-trained model also gives good results when the input images have a width of 352 pixels and a height of 288 pixels. Especially with video input, this reduces the computational complexity and shortens the time needed for inference.



---

## 6 Conclusion and outlook

Nano-backpack can be seen as an extension of the NAO robot, which increases the computing power of the system and at the same time enables an easy start into the development of further applications. Prospective users can use the provided webapp and should be able to let NAO walk and speak within minutes. This webapp and the corresponding Docker image can also be easily reused, modified or extended and thus be adapted to future requirements. For example the provided custom Python dataclass for depth estimation in videos could be combined with the *ALVideoDevice* service of the NAOqi-API. The entire architecture of the Nano-backpack enables the development in a modern Linux environment in which NAO-specific and many common software libraries are already available.

## References

- [1] M. Mattamala et al. "The NAO Backpack: An Open-hardware Add-on for Fast Software Development with the NAO Robot," arXiv preprint, 2017.
- [2] I. Alhashim and P. Wonka. "High Quality Monocular Depth Estimation via Transfer Learning," arXiv preprint, 2019.
- [3] National Bureau of Standards U.S. Department of Commerce Boulder "THERMAL CONDUCTIVITY OF ALUMINUM, COPPER, IRON, AND TUNGSTEN FOR TEMPERATURES FROM 1 K TO THE MELTING POINT", Colorado, June 1984.
- [4] Lakindu Akash, "linux-wifi-hotspot", 2019, <https://github.com/lakinduakash/linux-wifi-hotspot>
- [5] Sahiti Kappagantula, "Docker Explained – An Introductory Guide To Docker", 2020, <https://www.edureka.co/blog/docker-explained/>
- [6] Archlinux, "Dnsmasq", November 2020, <https://wiki.archlinux.org/index.php/Dnsmasq>

## Appendix A:



### DATA SHEET

## NVIDIA Jetson Nano System-on-Module

Maxwell GPU + ARM Cortex-A57 + 4GB LPDDR4 + 16GB eMMC

#### Maxwell GPU<sup>®</sup>

128-core GPU | End-to-end lossless compression | Tile Caching | OpenGL<sup>®</sup> 4.6 | OpenGL ES 3.2 | Vulkan<sup>™</sup> 1.1 | CUDA<sup>®</sup> | OpenGL ES Shader Performance (up to): 512 GFLOPS (FP16) | Maximum Operating Frequency: 921MHz

#### CPU

ARM<sup>®</sup> Cortex<sup>®</sup>-A57 MPCore (Quad-Core) Processor with NEON Technology | L1 Cache: 48KB L1 instruction cache (I-cache) per core; 32KB L1 data cache (D-cache) per core | L2 Unified Cache: 2MB | Maximum Operating Frequency: 1.43GHz

#### Audio

Industry standard High Definition Audio (HDA) controller provides a multichannel audio path to the HDMI interface.

#### Memory

Dual Channel | System MMU | Memory Type: 4ch x 16-bit LPDDR4 | Maximum Memory Bus Frequency: 1600MHz | Peak Bandwidth: 25.6 GB/s | Memory Capacity: 4GB

#### Storage

eMMC 5.1 Flash Storage | Bus Width: 8-bit | Maximum Bus Frequency: 200MHz (HS400) | Storage Capacity: 16GB

#### Boot Sources

eMMC and USB (recovery mode)

#### Networking

10/100/1000 BASE-T Ethernet | Media Access Controller (MAC)

#### Imaging

Dedicated RAW to YUV processing engines process up to 1400Mpix/s (up to 24MP sensor) | MIPI CSI 2.0 up to 1.5Gbps (per lane) | Support for x4 and x2 configurations (up to four active streams).

#### Operating Requirements

Temperature Range (T<sub>j</sub>): -25 – 97°C | Module Power: 5 – 10W | Power Input: 5.0V

#### Display Controller

Two independent display controllers support DSI, HDMI, DP, eDP: MIPI-DSI (1.5Gbps/lane); Single x2 lane | Maximum Resolution: 1920x960 at 60Hz (up to 24bpp) | HDMI 2.0a/b (up to 6Gbps) | DP 1.2a (HBR2 5.4 Gbps) | eDP 1.4 (HBR2 5.4Gbps) | Maximum Resolution (DP/eDP/HDMI): 3840 x 2160 at 60Hz (up to 24bpp)

#### Clocks

System clock: 38.4MHz | Sleep clock: 32.768kHz | Dynamic clock scaling and clock source selection

#### Multi-Stream HD Video and JPEG

##### Video Decode

H.265 (Main, Main 10): 2160p 60fps | 1080p 240fps  
H.264 (BP/MP/HP/Stereo SEI half-res): 2160p 60fps | 1080p 240fps

H.264 (MVC Stereo per view): 2160p 30fps | 1080p 120fps

VP9 (Profile 0, 8-bit): 2160p 60fps | 1080p 240fps

VP8: 2160p 60fps | 1080p 240fps

VC-1 (Simple, Main, Advanced): 1080p 120fps | 1080i 240fps

MPEG-2 (Main): 2160p 60fps | 1080p 240fps | 1080i 240fps

##### Video Encode

H.265: 2160p 30fps | 1080p 120fps

H.264 (BP/MP/HP): 2160p 30fps | 1080p 120fps

H.264 (MVC Stereo per view): 1440p 30fps | 1080p 60fps

VP8: 2160p 30fps | 1080p 120fps

JPEG (Decode and Encode): 600 MP/s

#### Peripheral Interfaces

xHCI host controller with integrated PHY: 1 x USB 3.0, 3 x USB 2.0 | USB 3.0 device controller with integrated PHY | EHCI controller with embedded hub for USB 2.0 | 4-lane PCIe: one x1/24 controller | single SD/MMC controller (supporting SDIO 4.0, SD HOST 4.0) | 3 x UART | 2 x SPI | 4 x I2C | 2 x I2S: support I2S, RJM, LJM, PCM, TDM (multi-slot mode) | GPIOs

#### Mechanical

Module Size: 69.6 mm x 45 mm | PCB: 8L HDI | Connector: 260 pin SO-DIMM

Note: Refer to the software release feature list for current software support; all features may not be available for a particular OS.

<sup>®</sup> Product is based on a published Khronos Specification and is expected to pass the Khronos Conformance Process. Current conformance status can be found at [www.khronos.org/conformance](http://www.khronos.org/conformance).

\* See the *Jetson Nano Thermal Design Guide* for details. Listed temperature range is based on module T<sub>j</sub> characterization.

## Appendix B:



### • AINOPE - Machen Sie Produkte und bieten Sie Dienstleistungen mit Herz

AINOPE widmet sich der Entwicklung, Herstellung und dem Verkauf von Peripheriegeräten für 3C- und Mobiltelefone. Verlassen Sie sich auf ein starkes Forschungs- und Entwicklungsteam und mehr als 15 Jahre harte Arbeit. Wir haben erfolgreich verschiedene heiße Produkte auf den Markt gebracht und mehr als 1000000 Käufer bedient. Viel Spaß beim Einkaufen mit AINOPE. Wir geben stets unser Bestes, um Ihnen ein hervorragendes Einkaufserlebnis zu bieten.

#### • Spezifikationen:

Kapazität: 5000mAh

Eingang: 5V/ 2A Micro USB / USB C

Ausgang: 5V/2A USB A

Abmessung: 3,7 x 2,3 x 0,5in / 94 x 58 x 13mm

Gewicht: 117 g / 4.12oz

Lieferumfang: 1 x 5000mAh Powerbank, 1 x Micro USB Kabel, 1 x User Handbuch

#### • Ladezustand Einfach zu Wissen:



Wenn 1%-25% Energie hat, leuchtet eine LED.

Wenn 25%-50% Energie hat, leuchten zwei LEDs.

Wenn 50%-75% Energie hat, leuchten drei LEDs.

Wenn 75%-100% Energie hat, leuchten vier LEDs.

## Appendix C:





### NVIDIA® Jetson Nano™ Thermal Transfer Plate

- Designed to fit the NVIDIA® Jetson Nano™ production modules
- NVIDIA® Jetson Nano™ production module does not include a thermal transfer plate
- CTI provides Thermal transfer plate that users can integrate into a custom chassis or to other interface material for further cooling
- Dimensions: 61mm x 40mm x 4.2mm

[Request Info/Quote](#)
[Download Manual](#)

[Download 3D Model](#)



Part #: XHG310

Description
Specifications
Ordering Information
Custom Design

#### Specifications

Compatibility	NVIDIA Jetson Nano
Dimensions	61mm x 40mm x 4.2mm
Weight	28g
Mounting Fasteners	M3.0X0.5, Ultra Low Profile 6.0mm long, Alloy Steel
Mounting Fasteners Torque Rating	3.1 in-lb