TECHNICAL DEEP DIVE

Anki Vector

A LOVE LETTER TO THE LITTLE DUDE

AUTHOR RANDALL MAAS

OVERVIEW This is fascicle explores how the Anki Vector was realized in hardware and software.

image by Step Dere

RANDALL MAAS has spent decades in Washington and Minnesota. He consults in embedded systems development, especially medical devices. Before that he did a lot of other things… like everyone else in the software industry. He is also interested in geophysical models, formal semantics, model theory and compilers.

You can contact him at randym@randym.name.

LinkedIn: http://www.linkedin.com/pub/randall-maas/9/838/8b1

Preface

The Anki Vector is a charming little robot – cute, playful, with a slightly mischievous character. It is everything I ever wanted to create in a bot. Sadly Anki went defunct shortly after releasing Vector.

This book is my attempt to understand the Anki Vector and its construction. The book is based on speculation. Speculation informed by Anki's SDKs, blog posts, patents and FCC filings; by articles about Anki, presentations by Anki employees; by PCB photos, and hardware teardowns from others; and by experience with the parts, and the functional areas.

1.1. CUSTOMIZATION AND PATCHING

What can be customized – or patched – in Vector?

- The firmware in the main processor may be customizable, that will be dicussed in the rest of the document
- The base-board is not field updatable, without expertise. The STM32F030 firmware can be extracted with a ST-Link ($25), and the the firmware can be disassembled — the microcontroller is conducive to this. Shy of recreating the firmware, the patches replace a key instruction here and there with a jump to the patch, created in assembly (most likely) code to fix or add feature, then jump back. (STM32's come with a boot loader, but the ST-Link is easier and inexpensive.)
- The cube firmware can be updated, but that seems both hard and not useful.

2. ORGANIZATION OF THIS DOCUMENT

- CHAPTER 1: PREFACE. This chapter describes the other chapters.
- CHAPTER 2: OVERVIEW OF VECTOR'S ARCHITECTURE. Introduces the overall design of the Anki Vector.

PART I: ELECTRICAL DESIGN.

- CHAPTER 3: VECTOR'S ELECTRICAL DESIGN. A detailed look at the electrical design of Vector.
- CHAPTER 4: ACCESSORY ELECTRICAL DESIGN. A look at the electrical design of Vectors accessories.

PART II: BASIC OPERATION.

- CHAPTER 5: ARCHITECTURE. A detailed look at Vectors overall software architecture.
- CHAPTER 7: COMMUNICATION. A look at the communication stack in Vector.
- CHAPTER 8: BLUETOOTH LE. The Bluetooth LE protocol that Vector responds to.
- CHAPTER 9: CLOUD. A look at the electrical design of Vectors accessories.

REFERENCES AND RESOURCES. This provides further reading and referenced documents.

APPENDICES: The appendices provides extra material

- APPENDIX A: ABBREVIATIONS, ACRONYMS, & GLOSSARY. This appendix provides a gloss of terms, abbreviations, and acronyms.
- APPENDIX B: TOOL CHAIN. This appendix lists the tools known or suspected to have been used by Anki to create, and customize the Vector, and for the servers. Tools that can be used to analyze vector
- APPENDIX C: BLUETOOTH LE PROTOCOLS. This appendix provides information on the Bluetooth LE interfaces to the companion Cube, and to Anki Vector
- APPENDIX D: SERVERS. This appendix provides the servers that the Anki Vector and App contacts
- APPENDIX E: PHRASES. This appendix reproduces the phrases that the Vector keys off of.
- APPENDIX F: FEATURES. This appendix enumerates the Vector OS "features" that can be enabled and disabled.
- APPENDIX G: FILE SYSTEM. This appendix lists the key files that are baked into the system.
- APPENDIX H: PLEO. This appendix gives a brief overview of the Pleo animatronic dinosaur.

Note: I use many diagrams from Cozmo literature. They're close enough

CHAPTER 2

Overview of Vector

Anki Vector is a cute, palm-sized robot; a buddy with a playful, slightly mischievous character. This chapter is provides an overview of Vector:

- Overview
- Ancestry: Cozmo

3. OVERVIEW

Vector is an emotionally expressive animatronic robot that we all love.

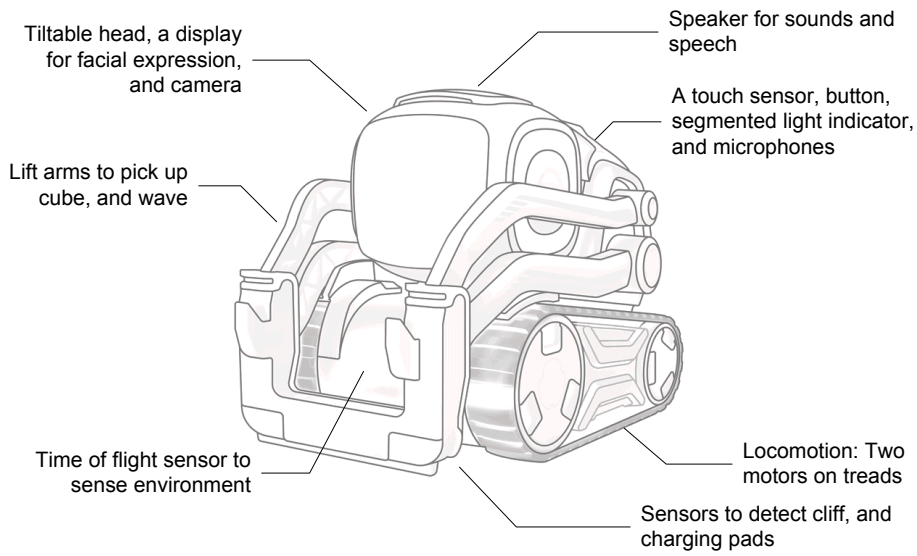


Figure 1: *Vector's main features*

He can express emotions thru expressive eyes (on an LCD display), raising and lower his head, sounds, wiggling his body (by using his treads), or lifting his arms… or shaking them.

Vector can sense surrounding environment, interact and respond to it. Recognize his name[1], follow the gaze of a person looking at him, and seek petting.[2]

3.1. FEATURES

Although cute, small, and affordable,[3] Vector's design is structured like many other robots.

[1] Vector can't be individually named.
[2] Admittedly this is a bit hit and miss.
[3] Although priced as an expensive toy, this feature set in a robot is usually an order of magnitude more, usually with less quality.

[3] Although priced as an expensive toy, this feature set in a robot is usually an order of magnitude more, usually with less quality.

He has a set of operator inputs:

- A touch sensor is used detect petting
- Internal microphone(s) to listen, hear commands and ambient activity level
- A button that is used to turn Vector on, to cause him to listen – or to be quiet (and not listen), to reset him (wiping out his personality and robot-specific information).
- He can detect his arms being raised and lowered.[4]

He has a set of indicators/annunciators:

- Segmented lights on Vectors backpack are used to indicate when he is on, needs the charger, has heard the wake word, is talking to the Cloud, can't detect WiFi, is booting, is resetting (wiping out his personality and robot-specific information).
- An LCD display, primarily to show eyes of a face. Robot eyes were Anki's strongest piece of imagery. Vector smiles and shows a range of expressions with his eyes.
- Speaker for cute sounds and speech synthesis

He has other means to express affect as well:

- His head can be tilted up and down to represent sadness, happiness, etc.
- His arms flail to represent frustration
- He can uses his treads to shake or wiggle, usually to express happiness or embarrassment

He has environmental sensors:

- A camera is used to map the area, detect and identify objects and faces.
- Fist-bump and being lifted can be detected using an internal IMU
- A forward facing "time of flight" proximity sensor aids in mapping and object avoidance
- Ground sensing proximity sensors that are used to detect cliffs and the edge of his area; these may also be used in following lines on his charger.

His internal sensing includes:

- Battery voltage, charging; charging temperature
- IMU for orientation position (6-axis)
- Encoders provide feedback on motor rotation

His other articulation & actuators are:

- Vector drives using two independent treads to do skid-steering
- Using his arms Vector can lift, or flip a cube; he can pop a wheelie, or lift himself over a small obstacle.
- Vector can raise and lower his head

Communication (other than user facing):

- Communication with the external world is thru WiFi and Bluetooth LE.
- Internally RS-232 (CMOS levels) and USB

[4] and possibly a pat on his head?

- The IMU and LCD are SPI attached
- The time of flight sensor connected via I2C

Motion control

- At the lowest level can control each of the motors speed, degree of rotation, etc. This allows Vector to make quick actions.
- Combined with the internal sensing, he can drive in a straight line and turn very tightly.
- Driving is done using a skid-steering, kinematic model
- To do all this, the motion control takes in feedback from the motor encoder, IMU-gyroscope. May also use the image processing for SLAM-based orientation and movement.

Guidance, path planning

- A*, Rapidly-Expanding Random Tree (RRT), D*-lite
- Paths are represented as arcs, line segments, and turn points

Navigation:

- Mocalization uses SLAM
- Mapping: uses IMU gyroscope, time of flight sensor, particle system. Representation: quad-tree (position, pose)

Behaviour system:

- Variety of behaviors animations
- Goals, linking up to the guidance system to accomplish them

Emotion model. Dimensions to emotional state

- happy (also referred to as his default state)
- confident
- social
- stimulated

Vision. Vision is Anki's hallmark: they used vision where others used beacons and mechanics

- Illumination sensing
- Motion sensing
- Links to Navigation system for mapping, (SLAM etc)
- Recognizing marker symbols in his environment
- Detecting faces and gaze detection allows him to maintain eye contanct

4. COZMO

We shouldn't discuss Vector without mentioning the prior generation. Vector's body is based heavily on Cozmo; the mechanical refinements and differences are relatively small. Nathaniel Monso's team designed Cozmos hardware. Vector's software architecture borrows from Cozmo…

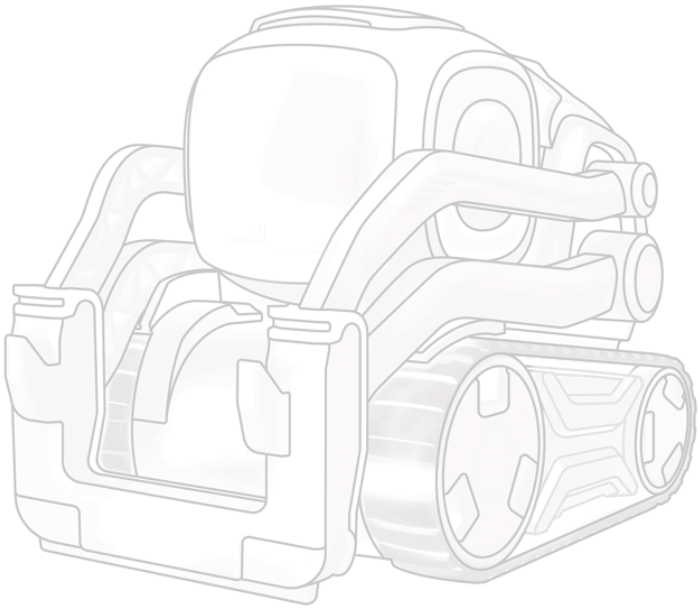
[This page is intentionally left blank for purposes of double-sided printing]

PART I

Electronics Design

This part provides an overview of the design of the electronics in Vector and his accessories

- VECTOR'S ELECTRICAL DESIGN. A detailed look at the electrical design of Vector.
- ACCESSORY ELECTRICAL DESIGN. A look at the electrical design of Vectors accessories.



[This page is intentionally left blank for purposes of double-sided printing]

CHAPTER 3

Electronics design description

This chapter describes the electronic design of the Anki Vector:

- Design Overview
- Detailed design of the main board
- Detailed design of the base-board
- Power characteristics

5. DESIGN OVERVIEW

Vector's design includes numerous some to sense and interact with his environment, other to interact with people and express emotion and behaviour.

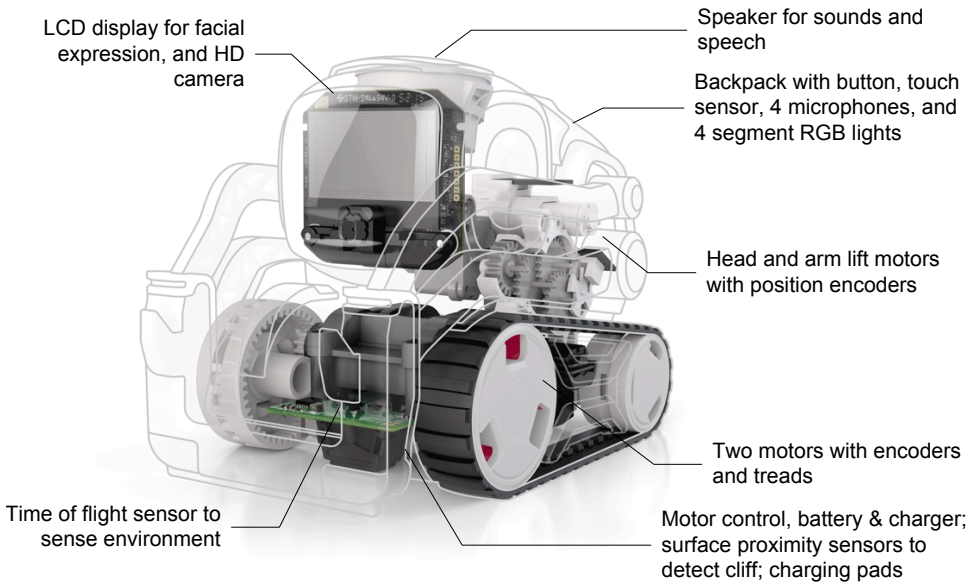


Figure 2: *Vector's main elements*

Vector's functional elements are:

Element	Description
backpack	The top of Vector, where he has a button, segmented lights, and a touch sensor.
battery	There is an internal battery pack (3.7v 320 mAh) that is used as Vector's source of energy.
button	A momentary push button is used to turn Vector on, to cause him to listen – or to be quiet (and not listen) – to reset him (wiping out his personality and robot-specific information).
camera	Vector uses an HD camera to visualize his environment, and recognize his human companions.
charging pad	Two pads on the bottom are used to replenish the energy in the battery pack from the dock.
LCD display	An IPS LCD, with an active area is 23.2mm x 12.1mm. It has a resolution of 184 x 96 pixels, with RGB565 color.
microphones	There are 4 internal microphone(s) to listen to commands and ambient activity level. Employs beam forming to localize sounds.
motors & encoders	There are four motors with single-step optical encoders to measure their position and approximate speed. One motor controls the tilt of the head assembly. Another controls the lift of his arms. Two are used to drive him in a skid-steering fashion.
segmented RGB lights	There are 4 LEDs used to indicate when he is on, needs the charger, has heard the wake word, is talking to the Cloud, can't detect WiFi, is booting, is resetting (wiping out his personality and robot-specific information).
speaker	A speaker is used to play sounds, and for speech synthesis
surface proximity sensors	4 infrared proximity sensors are used to detect the surface beneath Vector – and to detect drop offs ("cliffs") at the edge of his driving area.
time of flight sensor	A time of flight sensor is used to aid in mapping (by measuring distances) and object avoidance.
touch sensor	A touch allows Vector to detect petting and other attention.

Table 1: *Vectors main elements*

Vector has 6 circuit boards

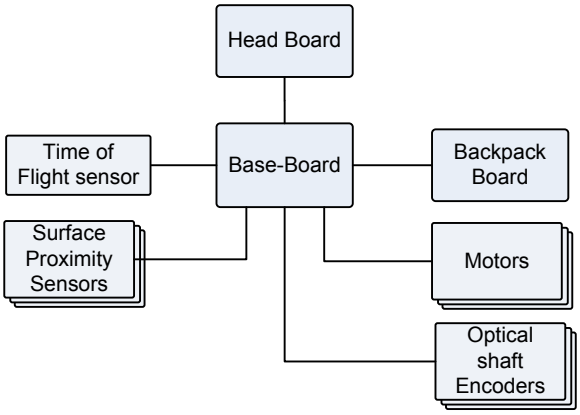


Figure 3: *Circuit board topology*

The main two boards are the head-board where the major of Vectors processing occurs, and the base-bard, which drives the motors and connects to the other boards.

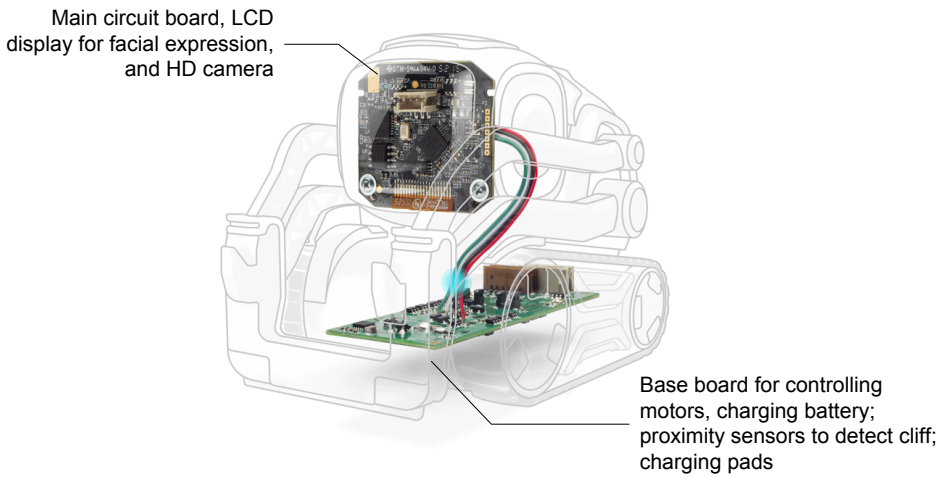


Figure 4: *Vector's main microcontroller circuit boards*

The table below summarizes the boards:

Circuit Board	Description
backpack board	The backpack board has 4 RGB LEDs, 4 MEMS microphones, a touch wire, and a button. This board connects to the base-board.
base-board	Drives the motor. power management battery charger
encoder-boards	The two encoder boards have single opto-coupler encoder each. The encoder is used to monitor the position of the arms and head, either as driven by the motor, or by a person manipulating them.
head-board	The head board includes the main processor, flash & RAM memory storage, an IMU, and a PMIC. The WiFi and Bluetooth LE are built into the processor. The camera and LCD are attached to the board, thru a flex tape. The speaker is also attached to this board.
time of flight sensor board	The time of flight sensor is on a separate board, allowing it to be mounted in Vectors front.

Table 2: *Vector's circuit boards*

5.1. POWER SOURCE AND DISTRIBUTION TREE

Vector is powered by a rechargeable battery pack, and the energy is distributed by the base-board:

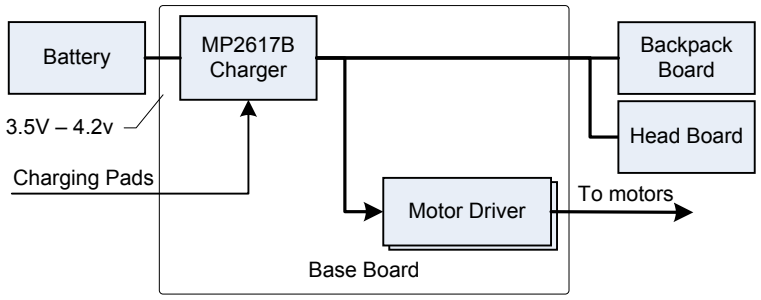


Figure 5: *Power distribution*

The MP2617B is a central element to managing the battery. It acts as a battery charger, a power switch and power converter for the whole system.

- When Vector is going into an *off* state – such as running too low on power, going into a ship state before first use, or has been turned off by a human companion – the MP2617B charger and power converted can be signaled to turn off
- When Vector is turned off the boards are not energized. The exception is that the high side of the push button is connected to the battery. When closed, the signals the MP2617B to connect the battery to the rest of the system, powering it up.
- The MP2617B is also responsible for charging the battery. There are two pads that mate the dock to supply energy to charge the battery.

In many rechargeable lithium ion battery systems there is a coulomb counter to track the state of charge. Vector does not have one. The need for recharge is triggered solely on the battery voltage.

Excessive current demand – such as from a stalled motor – can trigger a system brown-out and shutdown.

6. THE BACKPACK BOARD

The backpack board is effectively daughter board to the base-board. It provides extra IO and a couple of smart peripherals:

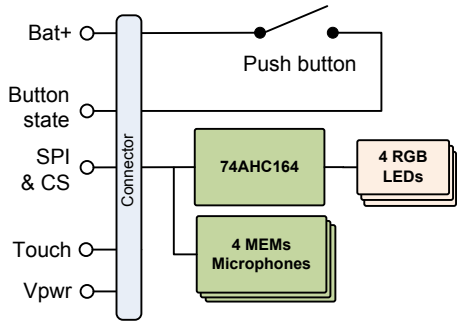


Figure 6: *Backpack board block diagram*

The table below summarizes the functional elements of the backpack board:

Table 3: *Backpack board functional elements*

Elements	Description
74AHC164	A SPI-based GPIO expander. This is used to drive the RGB LEDs.
microphones	There are 4 internal MEMS microphone(s). The microphones are accessed via SPI, in an output only mode. These are designated MK1, MK2, MK3, MK4
push button	A momentary push button is connected to the battery terminal, allowing a press to wake Vector, as well as signal the processor(s).
RGB LEDs	There are 4 RGB LEDs to make up a segmented display. Each segment can be illuminated individually, but may share a colour configuration with its counterparts.
touch sensor	A touch-sensing wire (and passive components)

6.1. BACKPACK CONNECTION

The backpack connection includes:

- Power and ground connections. This includes connection to the battery rail.
- The touch wire as an analog signal to the base-board
- A quasi digital signal out from the momentary push button
- (at least) Two chip selects
- A SPI-like set of clock, master-out-slave-in (MOSI) and *two* master-in-slave-out (MISO) signals

6.2. OPERATION

The touch sensor conditioning and sensing is handled by the base-board. The touch sense wire is merely an extension from the base-board through the backpack board.

The push-button is wired to the battery. When pressed, the other side of the push button signals both base-board microcontroller, and (if Vector is off) the charger chip to connect power. The theory of operation will be discussed further in the base-board section below.

The 74AHC164 serial-shift-register is used as a GPIO expander. It takes a chip select, clock signal and serial digital input, which are used to control up to 8 outputs. The inputs determine the state of 8 digital outputs used to control the RGB LEDs. More on this below.

Each of the 4 MEMS microphones take a chip select, clock signal, and provide a serial digital output. The clock signal (and one of the chip selects) is shared with the 74AHC164.

The base-board sets the digital outputs, and reads 2 microphones at a time. It reads all four microphones by alternating the chip selects to select which two are being accessed. (This will be discussed in the base-board section).

6.2.1 The LED control

Backpack LED control scheme

8 outputs are not enough to drive 4 RGB LEDs (each with 3 inputs) independently. 3 of the LEDs are always the same colour – but illuminated independently. The 4th LED may have a different colour, and is illuminated independently.

- D1 has separate red and green signals from the 74AHC164. It may share blue with the others.
- 3 signals from the 74AHC164 – Red, Green, and Blue – are shared for D2, D3, D4.[5]
- D2, D3, and D4 each have individual bottom drives

With care, the LEDs can be individually turned on and off (the low sides), and selected for a colour (the red, green, and blue signals).

[5] If I'm seeing the chip right, the ground, green and blue are wired together but that doesn't make sense in the truth-table to get the effect of the LED patterns

7. THE BASE-BOARD

The base board is a battery charger, smart IO expander, and motor controller. It connects the battery to the rest of the system, and is responsible for charging it. It is based on an STM32F030 which acts as second processor in the system.

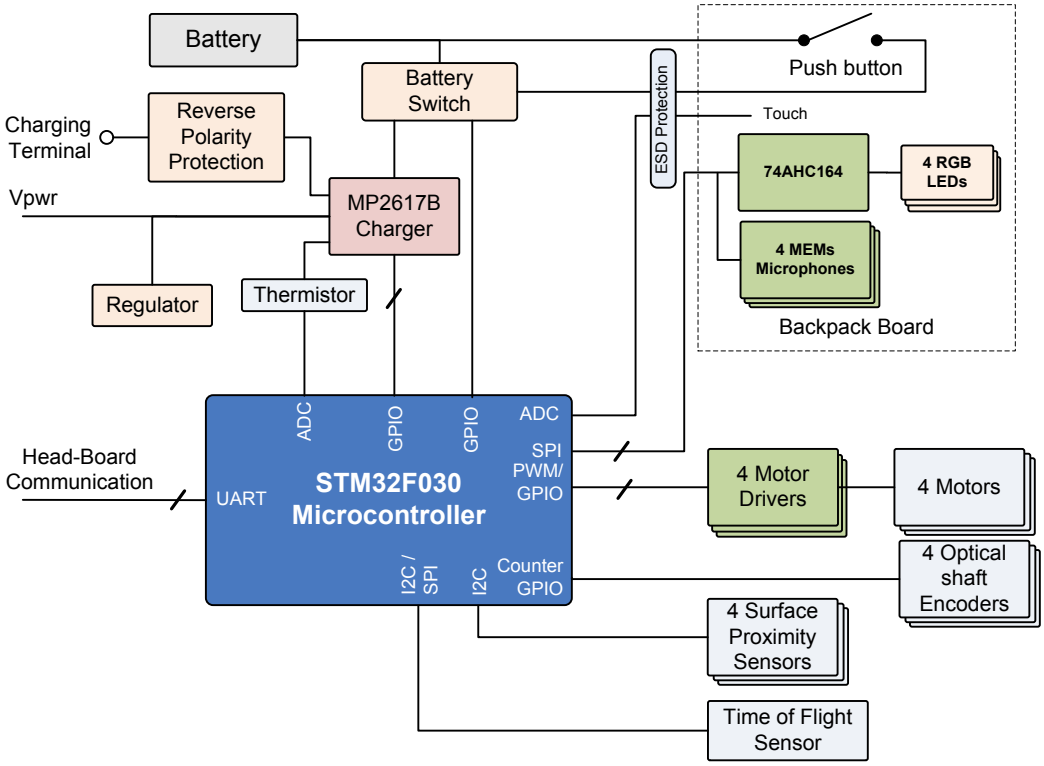


Figure 7: *Base-board block diagram*

The functional elements of the base-board are:

Table 4: *The base-board functional elements*

Element	Description
battery	An internal, rechargeable battery pack (3.7v 320 mAh)
battery switch	Used to disconnect the battery to support off-mode (such as when stored) and to reconnect the battery with a button press.
charging pad	Two pads on the bottom are used to replenish the energy in the battery pack from the dock.
motor driver	There are four motor drivers, based on an H-bridge design. This allows a motor to be driven forward and backward.
motors	There are four motors with to measure their position and approximate speed. One motor controls the tilt of the head assembly. Another controls the lift of his arms. Two are used to drive him in a skid-steering fashion.
MP2617B charger	The Monolithic Power Systems MP2617B serves as the battery charger. It provides a state of charge to the microcontroller.
optical shaft encoder	An opto-coupler, in conjunction with a slotted disc on a motor's shaft, which is used to measure a motors speed.
regulator	A 3.3v used to supply power to the microcontroller and logical components.
reverse polarity protection	Protects the circuitry from energy being applied to the charging pads in reverse polarity, such

	as putting Vector onto the charging pads in reverse.
STM32F030 microcontroller	The "brains" of the baseboard, used to drive the motors, and RGB LEDs; to sample the microphones, time of flight sensor, proximity sensor, temperature, and the touch sense;, and monitoring the battery charge state. It communicates with the head-board.
surface proximity sensors	4 infrared proximity sensors are used to detect the surface beneath Vector – and to detect drop offs ("cliffs") at the edge of his driving area.
thermistor[6]	A temperature sense resistor used measure the battery pack temperature; it is used to prevent overheating during recharge.
time of flight sensor	A time of flight sensor is used to measure distance to objects in front of Vector. This sensor is connected by I2C.

7.1. POWER MANAGEMENT

The battery charging is based on a MP2617B IC, which also provides some protection functions. There is no Coulomb counter; the state of charge is based solely on the battery voltage.

7.1.1 Battery pack

Vector's single-cell lithium battery is connected to the baseboard, and laid on top of the PCBA. The battery is not removable. The battery label has it as a 3.7v 320mAh pack. It is rechargeable. The pack is not a "smart" battery – it only has positive and negative leads, but lacks an onboard temperature sensor or BMS.

7.1.2 Protections

The charging pads have reverse polarity protection.

The MP2617B has an over-current cut off. If the current exceeds ~5A (4-6A), the battery will be disconnected from the system bus. Such a high-current indicates a short. There is no fuse.

The MP2617B has a low voltage cut off. If the battery voltage drops below ~2.4 (2.2-2.7V) the battery will be disconnected from the system bus (TBD) until the battery voltage rises above ~2.6V (2.4-2.8V).

The MP2617B has a temperature sense. If the temperature exceeds a threshold, charging is paused until the battery cools. The temperature sense is not on the battery. It is likely on the circuit board, or possibly top of the battery retention.

[6] Not identified. The customer service screen does show a battery pack temperature, indicating that this is reported.

7.1.3 Battery connect/disconnect

To preserve the battery need to isolate the battery from the rest of the system when in an off state. If there is minute current draw, the battery will irreversibly deplete while in storage even before the first sale. This constraint shapes the battery disconnect-reconnect logic. The schematic below shows one way to do this:

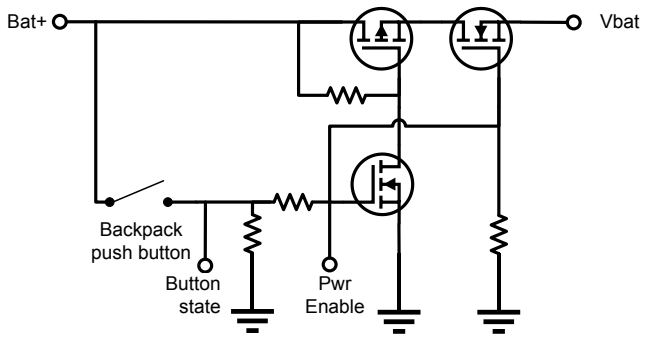


Figure 8: *A representative battery connect switch*

Two MOSFETS (a PFET and NFET)[7] act as a switch. These are in a single package, the DMC2038LVT. (This part is also used in the motor drivers.)

- When the system is in an off state, the MOSFETs are kept in an off state with biasing resistors. The PFET's gate is biased high with a resistor. The NFET gate is biased low, to ground. There is no current flow. Two MOSFETS are needed due to internal body diodes. The PFET body diode would allow current to flow from the battery (from the source to the drain). However this current is blocked by the NFET body diode, which has a different polarity
- The push button can wake the system. When the button is closed, the battery terminal (Bat+) is connected to the gate of the NFET, turning it on. A second NFET is also energized, pulling the PFET gate to ground, turning it on as well. When the button is open, Bat+ is not connected to anything, so there is no leakage path draining the battery.
- To keep the system energized when the button is open, the STM32F030 MCU must drive the Pwr Enable line high, which has the same effect as the button closed. The gate threshold voltage is 1V, well within the GPIO range of the MCU.
- The MCU can de-energize the system by pulling Pwr Enable line low. The switches will open, disconnect the battery.
- The MCU needs to be able to sense the state of the button while Pwr Enable is pulled high. The MCU can do this by sampling the Button State signal. This signal is isolated from from Pwr Enable by a large resistor, and pulled to ground by smaller resistor. This biases the signal to ground while the button is open.

This circuit also provides reverse polarity protection. It will not close the switch if the battery is connected backwards.

7.1.4 Charging

charging station pads

The charging station pads are connected to a MP2617B charger IC thru a reverse polarity protection circuit. The reverse polarity protection[8] is a DMG2305UX PFET in a diode

[7] Q11 and/or Q12

configuration. This approach has much lower losses than using an equivalent diode.

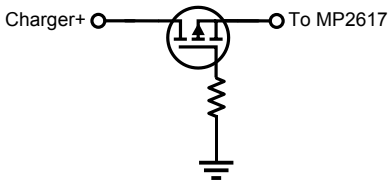


Figure 9: *A representative PFET based reversed polarity protection*

supplying power from the charging station

The MP2617B internally switches the charger input voltage to supply the system with power, and to begin charging the battery. This allows the charger to power the system even when the battery is depleted, or disconnected.

The presence of the dock power, and the state of MP2617B (charging or not) are signaled to the microcontroller.

charging states

The charger goes through different states as it charges the battery. Each state pulls a different amount of current from the charging pads and treats the battery differently.

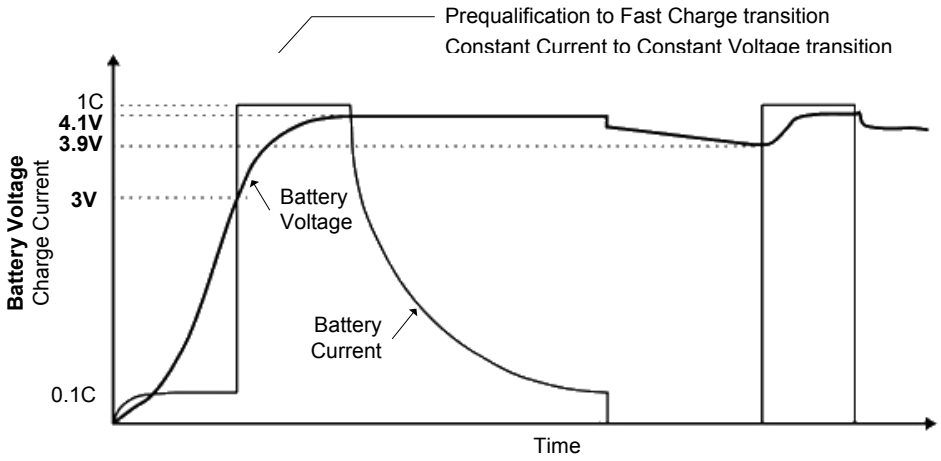


Figure 10: *Charging profile* (adapted from Texas Instruments)

constant current constant voltage

The basic idea is that the charger first applies a low current to the battery to bring it up to a threshold; this is called *prequalification* in the diagram. Then it applies a high current, call *constant current*. Once the battery voltage has risen to a threshold, the charger switches to *constant voltage,* and the current into the battery tapers off. I refer to the data sheet for more detail.

The MP2617B measures the battery temperature using a thermistor. If the temperature exceeds a threshold, charging is paused until the battery cools. The microcontroller also samples this temperature.

input current limits

The MP2617B supports limiting the input current, to accommodate the capabilities of external USB power converts. There are four different possible levels that the IC may be configured for: 2A is the default limit, 450mA to support USB2.0 limits, 825mA to support USB3.0 limits, and a custom limit that can be set by resistors. The input limit appears to be set for either default (up to ~2A input), or a programmable input.

Is the charger damaging the battery?

Commentary. In my testing, using a USB battery pack charging pulls up to 1A during the constant current, then falls off to 100mA-200mA during constant voltage, depending on the

[8] Q14

head-board's processing load. Stepped down to the ~4V battery the applied current at peak is approximately 1A.[9] This seems far too high.

Battery cells are normally charged at no more than a "1C" rate – in this case, the battery maximum charge rate should be 320mA at max. The IC data sheet supports a charging rate up to 2A.

My speculation is that, intentionally or unintentionally, the charger is configured for the default input limit of 2A and supports a faster charge. It is possible that the impact to battery life was considered low. My analysis could be wrong. As a preventative measure, I have a current limiter between my USB power adapter and Vector's charging dock.[10]

7.1.5 Brown-out

motor stall & brown out effects

The motor stall current is enough to cause Vector to brown-out and shut down unexpectedly. This indicates two possible mechanisms:

- If the system browns out the STM32F030, the MCU will no longer hold the power switch closed, and the system power will be disconnected.
- If the current exceeds a threshold, the MP2617B will disconnect power to the system. This threshold is very high – ~5A – and is unlikely to ever be encountered in operation.

Commentary: It may be interesting to modify either the MCU's Vdd to have a larger retaining capacitor, or to add a current limiting mechanism for the motors, such as an inline resistor.

7.2. ELECTRO-STATIC DISCHARGE (ESD) PROTECTION

The base-board employs a Vishay GMF05, TVS diode (U4) for electro-static discharge (ESD) protection, likely on the pushbutton and touch input.

7.3. STM32F030 MICROCONTROLLER

The base-board is controlled by a STM32F030C8 microcontroller (MCU). This processor essentially acts as a smart IO expander and motor controller.

The MCU's digital inputs:

- 4 optical shaft encoders, one for each motor (left, right, head, lift)
- Momentary push button
- 4 IR proximity sensor used to detect cliffs
- 2 charger state

The MCU's digital outputs:

- 4 motors enable
- 4 motors direction
- charger enable
- 3 chip selects

---

[9] Other reports suggest up to 2As into the battery, possible with the use of high-power USB adapters intended to support tablet recharge.

[10] 5Ω on the USB power. I tried 14ohms and 10 ohms; these should have limited the current to 360mA and 500mA respectively. Instead, Vector would only pull 20mA and 40mA; the later was enough to barely charge.

The MCU's analog inputs:

- Touch
- Battery voltage
- Temperature sensor (picks off the thermistor used by the MP2617)

The communication:

- 2 SPI, to LED outputs, from microphones. Uses an SPI MCLK to clock out the state, and MOSI to send the state of that IO channel
- I2C with time of flight
- UART

Note: The microcontroller does not have an external crystal[11] and uses an internal RC oscillator instead.

7.3.1 Manufacturing test connector

The base-board does include pads that appear to be intended for programming and test at manufacturing time.

7.4. SENSING

7.4.1 Time of Flight sensor

The MCU interfaces with a time of flight sensor, which can measure the distance to objects in front of vector. It "has a usable range 30mm to 1200mm away (max useful range closer to 300mm for Vector) with a field of view of 25 degrees."[12]

These sensors work by timing how long it takes for a coded pulse to return. The time value is then converted to a distance. Items too close return the pulse faster than the sensor can measure. The measured distance is available to the microcontroller over I2C.

7.4.2 Proximity sensing

Has 4 IR proximity sensors that are used to used to detect drops offs, or "cliffs." The exact model hasn't been identified, but the Everlight EAAPMST3923A2 is a typical proximity sensor. The sensor is an LED and IR detector pair. The sensor reports, via I2C, the brightness sensed by the detector. This are often pulsed, to reject to sunlight; and use a configurable threshold to reduce sensitivity to ambient light.

7.4.3 Touch sensing

The touch sensing works by alternating pulsing and sampling (with the ADC) the touch wire. The samples will vary "by various environmental factors such as whether the robot is on its charger, being held, humidity, etc."[13]

---

[11] as far as I can see
[12] Anki SDK, in the proximity.py file
[13] Anki SDK, in the touch.py file

7.5. MOTOR DRIVER AND CONTROL

Each motor driver is an H-bridge, allowing a brushed-DC motor to turn in either direction.

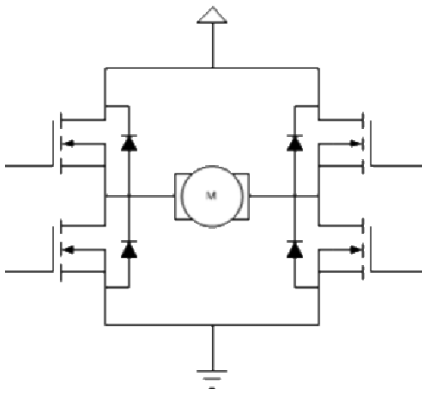


Figure 11: *Motor driver H-bridge*

Each side of the H-bridge based on the DMC2038LVT, which has a P-FET and N-FET in each package. Two of these are needed for each motor.

The MCU (probably) independently controls the high side and low side to prevent shoot thru. This is done by delaying a period of time between turning off a FET and turning on a FET.

The motors can controlled with a control loop that takes feedback from the optical encoder to represent speed and position.

7.6. COMMUNICATION

The base-board communicates with the head-board via RS-232 3.3V (estimated to be 2Mbps). As the MCU does not have a crystal, there may be communication issues from clock drift at extreme temperatures; since Vector is intended for use at room temperature, the effect may be negligible.

SPI communication with 2 microphones simultaneously

The communication with the backpack board is special. Two microphones are read at a time, using a shared SPI clock and chip select. The process can be:

1. The first chip select is asserted
2. A 16-bit SPI transfer is initiated on two SPI ports nearly simultaneously; the clock and data output (MOSI) on the second is ignored. This may be done carefully in code with as little as 1-instruction cycle skew.
    a. This transfer sends the state of the RGB LED's to the 74AHC164 chip
    b. The receiver accepts 16-bits each from the microphone 1 and 3.
3. After completion, the first chip select is de-asserted, and the second chip select is assert.
4. A 16-bit SPI transfer is initiated on two SPI ports nearly simultaneously; the clock and data output (MOSI) on the second is ignored. This transfers 16-bits each from the microphone 2 and 4.
5. After completion, the second chip select is de-asserted

The microphones are sampled a rate of 15625 samples/sec.

8. THE HEAD-BOARD (THE MAIN PROCESSOR BOARD)

The head-board handles the main processing. It is powered by a quad-core Arm-A7 Qualcomm APQ8009 microprocessor. The processor also connects to Bluetooth LE and WiFi transceivers, an HD camera, LCD display, speakers and an IMU.

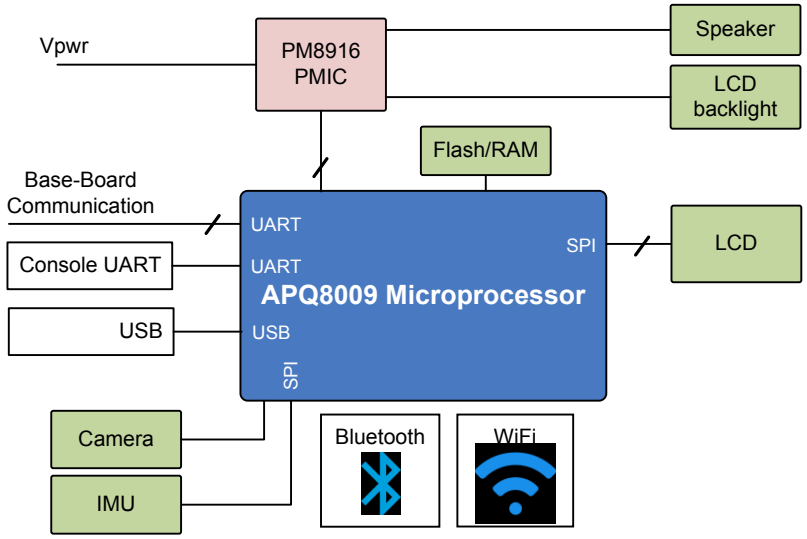


***Figure 12:** Head-board block diagram*

The head-board's functional elements are:

Element	Description
Bluetooth LE transceiver	A Bluetooth LE transceiver is built into the package
camera	Vector uses a 720P camera to visualize his environment, and recognize his human companions.
flash/RAM	Flash and RAM are provided by single external package, a Kingston 04EMCP04-NL3DM627 mixed memory chip with 4 GB flash and 512MB RAM.
inertial measurement unit (IMU)	The headboard includes a 6-axis IMU – gyroscope and accelerometer – used for navigation and motion control.
LCD display	An IPS LCD, with an active area is 23.2mm x 12.1mm. It has a resolution of 184 x 96 pixels, with RGB565 color.
microprocessor	The head-board is based on a Qualcomm APQ8009 (Snapdragon 212). processor is a quad-core Arm A7 (32-bit) CPU.
power management IC (PMIC)	The PM8916 power management IC provides voltage regulation for the processor, flash/RAM and other parts; it also provides audio out to the speaker, and controls the LCD backlight.
speaker	A speaker is used to play sounds, and for speech synthesis
WiFi transceiver	An 802.11AC WiFi transceiver is built into the processor package

***Table 5:** The head-boards functional elements*

8.1. SPEAKERS

The speaker is driven at 16bits, single channel, with a sample rate of 8000-16025 samples/sec.

8.2. CAMERA

Vector has a 720p with a 120° field of view. The camera is calibrated at manufacturing time.

The cameras power control is on GPIO #83

8.3. THE LCD

LCD display

Vector's LCD is a backlit IPS display. The processor is connected to the LCD via SPI. The backlight is PWM controlled by the PM8916 PMIC.

The prior generation, Cozmo, used an OLED display for his face and eyes. OLEDs are susceptible to burn-in and uneven dimming or discoloration of overused pixels. Anki addressed this with two accommodations. First it gave the eyes regular motion, looking around and blinking. Second, the LCD's illuminated rows were regularly alternated to give a retro-technology interlaced row effect, like old CRTs.

Vector's IPS display gives a smoother imagery that is much less susceptible to burnin, at the expense of higher power.

8.4. TRIM, CALIBRATION SERIAL NUMBERS AND KEYS

Each Vector has a set of per unit calibrations:

- The camera is calibrate
- The IMU is calibrated
- The motor power is calibrated[14]

There are per unit keys, MAC addresses and serial numbers

- Each processor has its own unique key, used to with the Trust Zone
- The WiFi and Bluetooth have assigned, unique MAC addresses.
- Each Vector has an assigned serial number

8.5. MANUFACTURING TEST CONNECTOR/INTERFACE

It is a common practice to include an interface on a product for use during manufacture. This is used to load firmware, unique ids – WiFi MACs, serial number –, to perform any calibration steps and to perform run-up checks that the device functions / is assembled correctly. It is intended to be a fast interface that doesn't cause yield fallout. Typically (but there are exception) this is not radio based, as they can interfere or have fiddly issues.

There are three possibilities

- There is a UART, that provides a boot console, but does not accept input
- There is a USB connector that probably is used to load firmware.
- The WiFi, once MAC addresses

The most likely test interface is the USB connection. Bluetooth LE is very unlikely due to the mismatch between its capabilities and the data rates necessary for video used to calibrate the camera.

[14] Todo: look into what this means. Is there a current sense for power? R43 looks like a sense resistor.. is that what it is for?

9. RESOURCES

Amitabha, *Benchmarking the battery voltage drain in Anki Vector and Cozmo*, 2018 Dec 31
https://medium.com/programming-robots/benchmarking-the-battery-voltage-drain-in-anki-vector-and-cozmo-239f23871bf8

Anki, *Lithium single-cell battery data sheet*
https://support.anki.com/hc/article_attachments/360018003653/Material%20Safety%20Data%20Sheet_April%202018.pdf

Diodes, Inc, *74AGC164 8-Bit Parallel-Out Serial Shift Registers*, Rev 2, 2015 Aug
https://www.diodes.com/assets/Datasheets/74AHC164.pdf

Diodes Inc, *DMG2305UX P-Channel Enhancement Mode MOSFET*
https://www.diodes.com/assets/Datasheets/DMG2305UX.pdf

Diodes, Inc, *DMC2038LVT Complementary Pair Enhancement Mode MOSFET*
https://www.diodes.com/assets/Datasheets/products_inactive_data/DMC2038LVT.pdf

Everlight *EAAPMST3923A2*

Kingston Technology, *Embedded Multi-Chip Package 04EMCP04-NL3DM627-Z02U*, rev 1.2, 2016
https://cdn.discordapp.com/attachments/573889163070537750/595223765206433792/04EMCP04-NL3DM627-Z02U_-_v1.2.pdf

Monolithic Power, *MP2617A, MP2617B 3A Switching Charger with NVDC Power Path* Management for Single Cell Li+ Battery, Rev 1.22 2017 Jun 29
https://www.monolithicpower.com/pub/media/document/MP2617A_MP2617B_r1.22.pdf

Panda, a data sheet for a similar single-cell lithium battery
https://panda-bg.com/datasheet/2408-363215-Battery-Cell-37V-320-mAh-Li-Po-303040.pdf

Qualcomm, *PM8916/PM8916-2 Power Management ICs Device Specification*, Rev C, 2018 Mar 13
https://developer.qualcomm.com/qfile/29367/lm80-p0436-35_c_pm8916pm8916_power_management_ics.pdf

ST Microelectronics, *STM32F030x8*, Rev 4, 2019-Jan
https://www.st.com/resource/en/datasheet/stm32f030c8.pdf

ST Microelectronics. Touch sensing
https://www.st.com/content/ccc/resource/technical/document/application_note/group0/ed/0d/4d/87/04/1d/45/e5/DM00445657/files/DM00445657.pdf/jcr:content/translations/en.DM00445657.pdf
https://www.st.com/en/embedded-software/32f0-touch-lib.html
https://hsel.co.uk/2016/05/22/stm32f0-software-capacitive-touch/
https://github.com/pyrohaz/STM32F0-SoftTouch

CHAPTER 4

Accessory Electronics design description

This chapter describes the electronic design of the Anki Vector accessories:

- The charging station
- The companion cube

10. CHARGING STATION

The charging station is intended to provide energy to the Vector, allowing it to recharge.

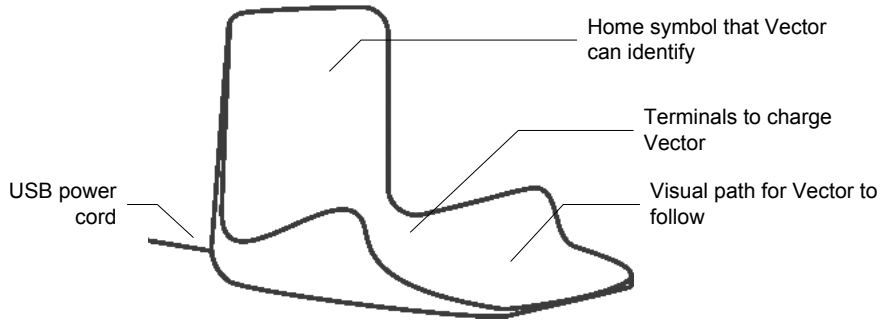


Figure 13: *Charging station main features*

The charging station has a USB cable that plugs into an outlet adapter or battery. The adapter or battery supplies power to the charging station. The base of the station has two terminals to supply +5V (from the power adapter) to Vector, allowing him to recharge. The terminals are offset in such a way to prevent Vector from accidentally being subject to the wrong polarity. Vector has to be backed into charging station in mate with the connectors. Vector face-first, even with his arms lifted, will not contact the terminals.

The charging station has an optical marker used by Vector to identify the charging station and its pose (see chapter TBD).

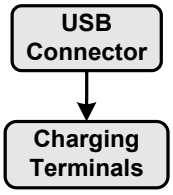


Figure 14: *Charging station block diagram*

11. CUBE

This section describes the companion cube accessory. The companion cube is a small toy for Vector play with. He can fetch it, roll it, and use it to pop-wheelies. Each face of the cube has a unique optical marker used by Vector to identify the cube and its pose (see chapter TBD).

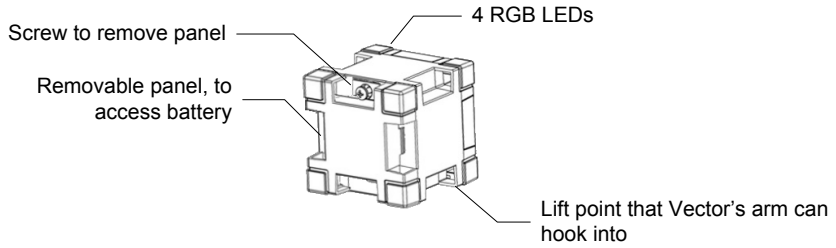


Figure 15: *Cube's main features*

Although the companion cube is powered, this is not used for localization or pose. The electronics are only used to flash lights for his owner, and to detect when a person taps, moves the cube or changes the orientation.

The cube has holes near the corners to allow the lift to engage, allowing Vector to lift the cube. Not all corners have such holes. The top – the side with the multicolour LEDs – does not have these. Vector is able to recognize the cubes orientation by symbols on each face, and to flip the cube so that it can lift it.

The electronics in the cube are conventional for a small Bluetooth LE accessory:

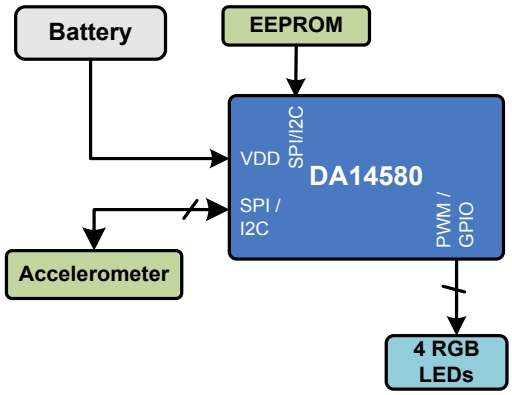


Figure 16: *Block diagram of the Cube's electronics*

The Cube's electronic design includes the following elements:

Element	Description
accelerometer	Used to detect movement and taps of the cube.
battery	The cube is powered by a 1.5 volt N / E90 / LR1 battery cell.[15]
crystal	The crystal provides the accurate frequency reference used by the Bluetooth LE radio.
Dialog DA14580	This is the Bluetooth LE module (transmitter/receiver, as well as microcontroller and protocol implementation).
EEPROM	The EEPROM holds the updatable application firmware.
RGB LEDs	There are 4 RGB LEDs. They can flash and blink. Unlike the backpack LEDs, two LEDs can

Table 6: *The Cube's electronic design elements*

[15] The size is similar to the A23 battery, which will damage the cubes electronics.

have independent colors.

The communication protocol is given appendix C.

11.1. OVER THE AIR FIELD UPDATES

The DA14580 has a minimal ROM bootloader that initializes hardware, moves a secondary bootloader from “One Time Programmable” ROM (OTP) into SRAM, before passing control to it. The firmware is executed from SRAM to reduce power consumption. The secondary bootloader loads the application firmware from I2C or SPI EEPROM or flash to SRAM and pass control to it.

If the application passes control back to the bootloader – or there isn't a valid application in EEPROM –a new application can be downloaded. The bootloader uses a different set of services and characteristics to support the bootloading process.

11.2. RESOURCES

Dialog, *SmartBond™ DA14580 and DA14583*
https://www.dialog-semiconductor.com/products/connectivity/bluetooth-low-energy/smartbond-da14580-and-da14583

Dialog, *DA14580 Low Power Bluetooth Smart SoC, v3.1, 2015 Jan 29*

Dialog, *UM-B-012 User manual DA14580/581/583 Creation of a secondary bootloader*, CFR0012-00 Rev 2, 2016 Aug 24
Dialog, *Application note: DA1458x using SUOTA,* AN-B-10, Rev 1, 2016-Dec-2
https://www.dialog-semiconductor.com/sites/default/files/an-b-010_da14580_using_suota_0.pdf

[This page is intentionally left blank for purposes of double-sided printing]

PART II

Basic Operation

This part provides an overview of the design of the electronics in Vector and his accessories

- THE MAIN SOFTWARE MODULES. A detailed look at Vectors overall software architecture.
- COMMUNICATION. A look at the communication stack in Vector.
- BLUETOOTH LE. The Bluetooth LE protocol that Vector responds to.
- CLOUD. A look at how Vector syncs with the cloud

[This page is intentionally left blank for purposes of double-sided printing]

CHAPTER 7

Communication

This chapter is about the communication system:

- Internal communication with the base-board, and internal peripherals
- Bluetooth LE: with the Cube, and with the application
- WiFi: with the cloud, and with the application
- Internal support

12. OVERVIEW OF VECTOR'S COMMUNICATION INFRASTRUCTURE

A significant part of Vector's software is focused on communication.

- Internal IPC between processes
- Communication with local peripherals and the base-board processor
- Communication with external accessories and applications.

The rough stack:

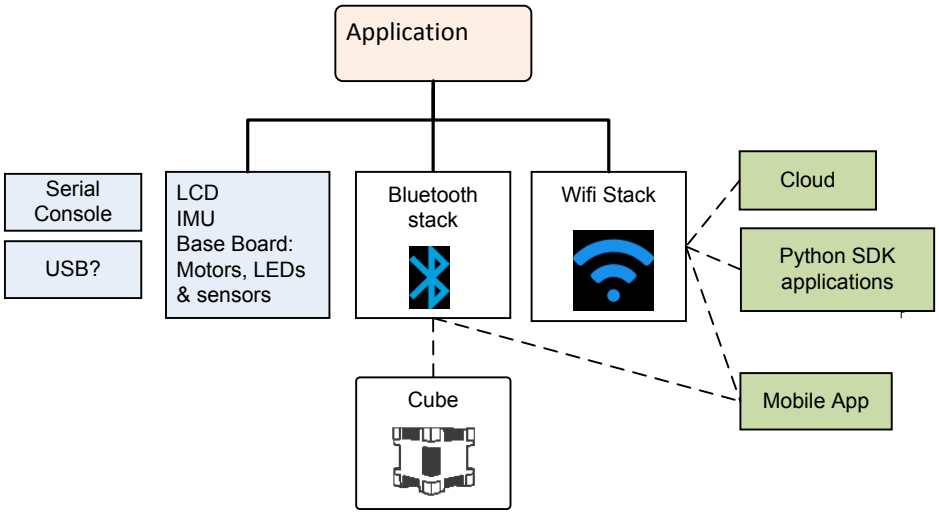


Figure 17: *The overall communication infrastructure*

13. INTERNAL COMMUNICATION WITH PERIPHERALS

Communication stack within the software. One part Linux, one part Qualcomm, and a big heaping dose of Anki's stuff.

13.1. COMMUNICATION WITH THE BASE-BOARD

The head board communicates with the base board using a serial interface[16]

Data rate: estimated to be ~2Mbps

Messages from Base to Head are a regular, fixed-size packet, containing:

- The state of the backpack button
- The touch sensor voltage
- The microphone signals for all 4 microphones. (This could be 16 bits, or signed 8 bit for delta-sigma changes.)
- The battery voltage
- State of the charger (on dock/etc)
- The temperature of the battery or charger
- The state of 4 motor encoders, possibly as encoder counters, possibly as IO state
- The time of flight reading, probably 16bits in mm
- The voltage (or other signal) of each of the 4 cliff proximity sensors

The messages from the head board to the base-board have the content:

- The 4 LED RGB states
- Controls for the motors: possible direction and enable; direction and duty cycle; or a target position and speed.
- Power control information: disable power to the system, turn off distance, cliff sensors, etc.

13.2. SERIAL BOOT CONSOLE

The head-board includes a 115200, 8 data bits no parity, 1 stop bit; the device file is /dev/ttyHS0. Only prints the boot console. (This is passed in the commanded line by the bootloader)

13.3. USB

There are pins for USB on the head board. Asserting "F_USB" pad to VCC enables the port. During power-on, and initial boot it is a Qualcomm QDL port. The USB supports a Qualcomm debugging driver (QDL), but the readout is locked. It appears to be intended to inject firmware during manufacture.

The /etc/initscripts/usb file enables the USB and the usual functionfs adb. It lines in /sbin/usr/composition/9091 (I think, if I understand the part number matching correctly). This launches ADB (DIAG + MODEM + QMI_RMNET + ADB)

Melanie_t reports this not working, not enabled.

14. BLUETOOTH LE

Bluetooth LE is used for two purposes:

1. Bluetooth LE is used to initially configure Vector, to reconfigure him when the Wifi changes; to allow him to interact with the cube. Potentially allows some diagnostic and customization.

[16] /dev/tty unknown

2. Bluetooth LE is used to communicate with the companion Cube accessory: to detect its movement, taps, and to set the state of its LEDs.

Vector's Bluetooth LE stack looks like:

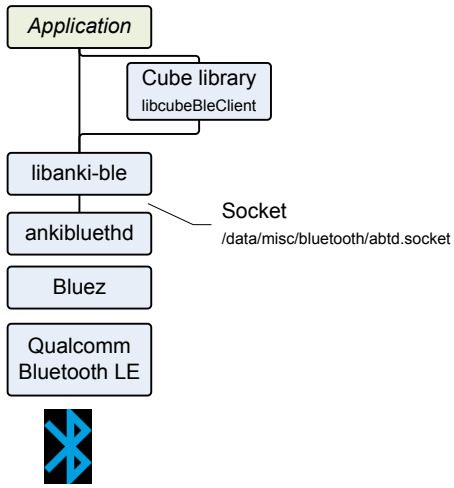


Figure 18: *The Bluetooth LE stack*

The elements of the Bluetooth LE stack include:

Element	Description & Notes
ankibluetoothd	A server daemon. The application layer communicates with it over a socket; `/data/misc/bluetooth/abtd.socket`
bccmd	A Bluetooth core command
btmon	A command-line Bluetooth tool
libanki-ble.so	Communicates with Anki Bluetooth daemon probably serves both the external mobile application interface and communication with the cube
libcubeBleClient.so[17]	A library to communicate with Cube, play animations on its LEDs, detect taps and orientations.
viccubetool	Probably used to update the firmware in the Cube.

Table 7: *Elements of the Bluetooth LE stack*

[17] The library includes great deal of built in knowledge of the state of application ("game engine"), animations, and other elements

CHAPTER 8

Bluetooth LE Communication Protocol

This chapter is describes Vector's Bluetooth LE communication protocol.

- The kinds of activities that can be done thru communication channels
- The interaction sequences
- The communication protocol stack, including encryption, fragmentation and reassembly.

Note: communication with the Cube is simple reading and writing a characteristic, and covered in Appendix C.

15. COMMUNICATION PROTOCOL OVERVIEW

Communication with Vector, once established, is structure as a request-response protocol. The request and responses are referred to as "C-Like Abstract Data structures" (CLAD) which are fields and values in a defined format, and interpretation. Several of these messages are used to maintain the link, setting up an encryption over the channel.

The application layer messages may be arbitrarily large. To support Bluetooth LE 4.1 (the version in Vector, and many mobile devices) the CLAD message must be broken up into small chunks to be sent, and then reassembled on receipt.

Combined with application-level encryption, the communication stack looks like:

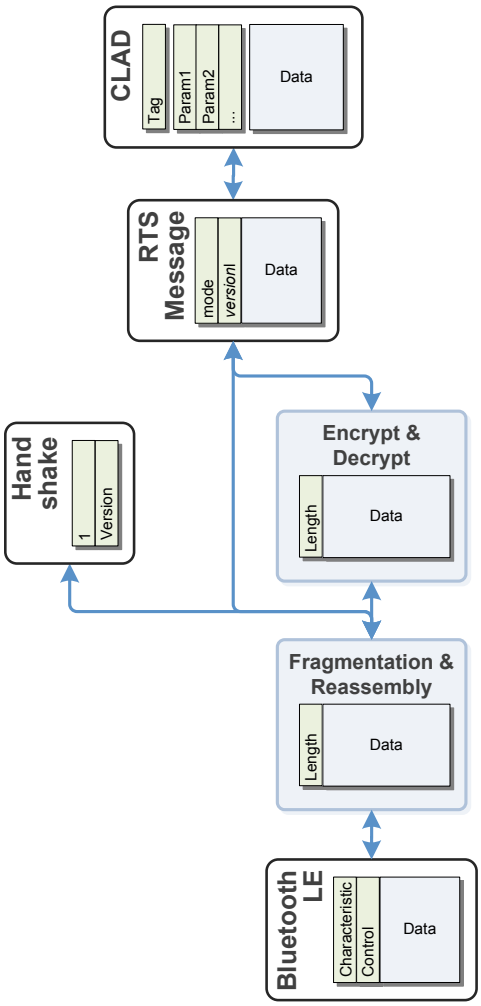


Figure 19: *Overview of encryption and fragmentation stack*

THE BLUETOOTH LE is the link/transport media. It handles the delivery, and low-level error detection of exchanging message frames. The frames are fragments of the overall message. The GUID's for the services and characteristics can be found in Appendix C.

THE FRAGMENTATION & REASSEMBLY is responsible for breaking up a message into multiple frames, and reassembling them into a message.

THE ENCRYPTION & DECRYPTION LAYER is used to encrypt and decrypt the messages, after the communication channel has been set up.

THE RTS is extra framing information that identifies the kind of CLAD message, and the version of its format. The format changed with version, so this version code is embedded at this layer.

THE C-LIKE ABSTRACT DATA (CLAD) is the layer that decodes the messages into values for fields, and interprets them,

15.1. SETTING UP THE COMMUNICATION CHANNEL

It sometimes helps to start with the over all process. This section will walk thru the process, referring to later sections where detailed information resides.

If you use "first time" – or wish to re-pair with him – put him on the charger, and press the backpack button twice quickly. He'll display a screen indicating he is getting ready to pair.

If you have already paired the application with Vector, the encryption keys can be reused.

1. The application opens Bluetooth LE connection (retrieving the service and characteristics handles), and subscribes to the "read" characteristic (see Appendix C for the UUID).
2. Vector sends *handshake* message; which the application receives. The handshake message structure is given below. The handshake message includes the version of the protocol supported.

Table 8: *Parameters for Handshake message*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*type*	?
1	4	uint32_t	*version*	The version of the protocol/messages to employ

3. The application sends the handshake back
4. Then the Vector will send a *connection request,* consisting of the public key to use for the session. The application's response depends on whether this is a first time pairing, or a reuse.
    a. First time pairing requires that Vector have already been placed into pairing mode prior to connecting to Vector. The application keys should be created (see section *15.3.1 First time pairing* above).
    b. Reconnection can reuse the public and secret keys, and the encryption and decryption keys from a prior pairing
5. The application should then send the publicKey in the response
6. If this is a first time pairing, Vector will display a *pin code*. This is used to create the public and secret keys, and the encryption and decryption keys (see section *15.3.1 First time pairing* above). These can be saved for use in future reconnection.
7. Vector will send a *nonce* message. After the application has sent its response, the channel will now be encrypted.
8. Vector will send a *challenge* message. The application should increment the passed value and send it back as a challenge message.
9. Vector will send a *challenge success* message.
10. The application can now send other commands

If the user puts Vector on the charger, and double clicks the backpack button, Vector will usually send a *disconnect* request.

15.2. FRAGMENTATION AND REASSEMBLY

An individual frame sent over Bluetooth LE is limited to 20 bytes. (This preserves compatibility with Bluetooth LE 4.1) A frame looks like:

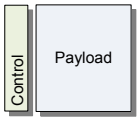


Figure 20: *The format of a frame*

The control byte is used to tell the receiver how to *reassemble* the message using this frame.

- If the MSB bit (bit 7) is set, this is the start of a new message. The previous message should be discarded.
- If the 2nd MSB (bit 6) is set, this is the end of the message; there are no more frames.
- The 6 LSB bits (bits 0..5) are the number of payload bytes in the frame to use.

The receiver would append the payload onto the end of the message buffer. If there are no more frames to be received it will pass the buffer (and size count) on to the next stage. If encryption has been set up, the message buffer will be decrypted and then passed to the RTS and CLAD. If encryption has not been set up, it is passed directly to the RTS & CLAD.

Fragmenting reverses the process:

1. Set the MSB bit of the control byte, since this is the start of a message.
2. Copy up to 19 bytes to the payload.
3. Set the number of bytes in the 6 LSB bits of the control byte
4. If there are no more bytes remaining, set the 2nd MSB it of the control byte.
5. Send the frame to Vector
6. If there are byte remaining, repeat from step 2.

15.3. ENCRYPTION SUPPORT

For the security layer, you will need the following

```
uint8_t Vectors_publicKey[32];
uint8_t publicKey [crypto_kx_PUBLICKEYBYTES];
uint8_t secretKey [crypto_kx_SECRETKEYBYTES];
uint8_t encryptionKey[crypto_kx_SESSIONKEYBYTES];
uint8_t decryptionKey[crypto_kx_SESSIONKEYBYTES];
uint8_t encryptionNonce[24];
uint8_t decryptionNonce[24];
uint8_t pinCode[16];
```

The variables mean:

Table 9: *The encryption variables*

Variable	Description
decryptionKey	The key used to decrypt each message from to Vector.
decryptionNonce	An extra bit that is added to each message. The initial nonce's to use are provided by Vector.
encryptionKey	The key used to encrypt each message sent to Vector.
encryptionNonce	An extra bit that is added to each message as it is encrypted. The initial nonce's to use are provided by Vector.
pinCode	6 digits that are displayed by Vector during an initial pairing.
Vectors_publicKey	The public key provided by Vector, used to create the encryption and decryption keys.

There are two different paths to setting up the encryption keys:

- First time pairing, and
- Reconnection

15.3.1 First time pairing

First time pairing requires that Vector be placed into pairing mode prior to the start of communication. This is done by placing Vector on the charger, and quickly double clicking the backpack button.

The application should generate its own internal *public* and *secret keys* at start.

```
crypto_kx_keypair(publicKey, secretKey);
```

The application will send a *connection response* with first-time-pairing set, and the public key. After Vector receives the connection response, he will display the *pin code*. (See the steps in the next section for when this will occur.)

The session *encryption* and *decryption keys* can then created:

```
crypto_kx_client_session_keys(decryptionKey, encryptionKey, publicKey, secretKey,
      Vector_publicKey);
size_t pin_length = strlen(pin);

crypto_generichash(encryptionKey, sizeof(encryptionKey), encryptionKey,
      sizeof(encryptionKey), pin, pin_length);
crypto_generichash(decryptionKey, sizeof(decryptionKey), decryptionKey,
      sizeof(decryptionKey), pin, pin_length);
```

15.3.2 Reconnecting

Reconnecting can reused the public and secret keys, and the encryption and decryption keys. It is not known how long these persist on Vector. {Next pairing? Next reboot? Indefinitely?}

15.3.3 Encrypting and decryption messages

Vector will send a *nonce* message with the *encryption* and *decryption nonces* to employ in encrypting and decrypting message.

Each received enciphered message can be decrypted from cipher text (cipher, and cipherLen) to the message buffer (message and messageLen) for further processing:

```
crypto_aead_xchacha20poly1305_ietf_decrypt(message, &messageLen, NULL, cipher,
      cipherLen,  NULL, 0L, decryptionNonce, decryptionKey);
sodium_increment(decryptionNonce, sizeof decryptionNonce);
```

Note: the decryptionNonce is incremented each time a message is decrypted.

Each message to be sent can be encrypted from message buffer (message and messageLen) into cipher text (cipher, and cipherLen) that can be fragmented and sent:

```
crypto_aead_xchacha20poly1305_ietf_encrypt(cipher, &cipherLen, message, messageLen,
      NULL, 0L, NULL, encryptionNonce, encryptionKey);
sodium_increment(encryptionNonce, sizeof encryptionNonce);
```

Note: the encryptionNonce is incremented each time a message is encrypted.

15.4. THE RTS LAYER

There is an extra, pragmatic layer before the messages can be interpreted by the application. The message has two to three bytes at the header:

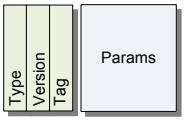


Figure 21: *The format of an RTS frame*

- The type byte is either 1 or 4. If it is 1 the version of the message format is 1.
- If type byte is 4, the version is held in the next byte. (If the type is 1, there is no version byte).
- The next byte is the tag – the value used to interpret the message.

The tag, parameter body, and version are passed to the CLAD layer for interpretation. This is described in the next section.

16. MESSAGE FORMATS

This section describes the format and interpretation of the CLAD messages that go between the App and Vector. It describes the fields and how they are encoded, etc. Fields that do not have a fixed location, have no value for their offset. Some fields are only present in later versions of the protocol. They are marked with the version that they are present.

Except where otherwise stated:

- Requests are from the mobile application to Vector, and responses are Vector to the application
- All values in little endian order

Table 10: *Summary of the commands*

	Request	Response	Min Version
Application connection id	$1F_{16}$	20_{16}	4
Cancel pairing	10_{16}		0
Challenge	04_{16}	04_{16}	0
Challenge success	05_{16}		0
Connect	01_{16}	02_{16}	0
Cloud session	$1D_{16}$	$1E_{16}$	3
Disconnect	11_{16}		0
File download	26_{16}		2
Log	18_{16}	19_{16}	2
Nonce	03_{16}	12_{16}	
OTA cancel	17_{16}		2
OTA update	$0E_{16}$	$0F_{16}$	0
SDK proxy	22_{16}	23_{16}	5
Response	21_{16}		4
SSH	15_{16}	16_{16}	0
Status	$0A_{16}$	$0B_{16}$	0
WiFi access point	13_{16}	14_{16}	0
WiFi connect	06_{16}	07_{16}	0
WiFi forget	$1B_{16}$	$1C_{16}$	3
WiFi IP	08_{16}	09_{16}	0
WiFi scan	$0C_{16}$	$0D_{16}$	0

16.1. APPLICATION CONNECTION ID

?

16.1.1 Request

The parameters of the request body are:

Offset	Size	Type	Parameter	Description
0	2	uint16_t	*name length*	The length of the application connection id; may be 0
2	*varies*	uint8_t[name length]	*name*	The application connection id

Table 11: *Parameters for Application Connection Id request*

16.1.2 Response

There is no response.

16.2. CANCEL PAIRING

Speculation: this is sent by the application to cancel the pairing process

16.2.1 Request

The command has no parameters.

16.2.2 Response

There is no response.

16.3. CHALLENGE

This is sent by Vector if he liked the response to a nonce message.

16.3.1 Request

The parameters of the request body are:

Offset	Size	Type	Parameter	Description
0	4	uint8_t	*value*	The challenge value

Table 12: *Parameters for challenge request*

The application, when it receives this message, should increment the value and send the response (a challenge message).

16.3.2 Response

The parameters of the response body are:

Offset	Size	Type	Parameter	Description
0	4	uint8_t	*value*	The challenge value; this is 1 + the value that was received.

Table 13: *Parameters for challenge response*

If Vector accepts the response, he will send a *challenge success.*

16.4. CHALLENGE SUCCESS

This is sent by Vector if the challenge response was accepted.

16.4.1 Request

The command has no parameters.

16.4.2 Response

There is no response.

16.5. CLOUD SESSION

This command is used to request a cloud session [TBD]

16.5.1 Command

The parameters of the request body are:

Offset	Size	Type	Parameter	Description
0	2	uint16_t	*token length*	The number of bytes in the token; may be 0
2	varies	uint8_t	*token*	The session token
	1	uint8_t	*client name length*	The number of bytes in the client name string; may be 0 version >= 5
	varies	uint8_t[]	*client name*	The client name [?] string version >= 5
	1	uint8_t	*application id length*	The number of bytes in the application id string; may be 0 version >= 5
	varies	uint8_t[]	*application id*	The application id version >= 5

Table 14: *Parameters for Cloud Session request*

16.5.2 Response result

The parameters for the connection response message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*success*	0 if failed, otherwise successful
1	1	uint8_t	*status*	See *Table 16: Cloud status enumeration*
2	1	uint16_t	*token length*	The number of bytes in the client token GUID; may be 0
	varies	uint8_t[]	*token*	The client token GUID

Table 15: *Parameters for Cloud Session Response*

The cloud status types are:

Index	Meaning
0	unknown error
1	connection error
2	wrong account
3	invalid session token
4	authorized as primary
5	authorized as secondary
6	reauthorization

Table 16: *Cloud status enumeration*

16.6. CONNECT

The connect request *comes from Vector* at the start of a connection. The response is from the application.

16.6.1 Request

The parameters of the request body are:

Offset	Size	Type	Parameter	Description
0	32	uint8_t[32]	*publicKey*	The public key for the connection

Table 17: *Parameters for Connection request*

The application, when it receives this message, should use the public key for the session, and send a response back.

16.6.2 Response

The parameters for the connection response message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*connectionType*	See *Table 19: Connection types enumeration*
1	32	uint8_t[32]	*publicKey*	The public key to use for the connection

Table 18: *Parameters for Connection Response*

The connection types are:

Index	Meaning
0	first time pairing (requests pin code to be displayed)
1	reconnection

Table 19: *Connection types enumeration*

The application sends the response, with its publicKey (see section *15.2 Fragmentation and* reassembly

An individual frame sent over Bluetooth LE is limited to 20 bytes. (This preserves compatibility with Bluetooth LE 4.1) A frame looks like:

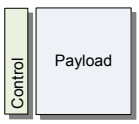


Figure 20: *The format of a frame*

The control byte is used to tell the receiver how to *reassemble* the message using this frame.

- If the MSB bit (bit 7) is set, this is the start of a new message. The previous message should be discarded.
- If the 2nd MSB (bit 6) is set, this is the end of the message; there are no more frames.
- The 6 LSB bits (bits 0..5) are the number of payload bytes in the frame to use.

The receiver would append the payload onto the end of the message buffer. If there are no more frames to be received it will pass the buffer (and size count) on to the next stage. If encryption has been set up, the message buffer will be decrypted and then passed to the RTS and CLAD. If encryption has not been set up, it is passed directly to the RTS & CLAD.

Fragmenting reverses the process:

7. Set the MSB bit of the control byte, since this is the start of a message.
8. Copy up to 19 bytes to the payload.
9. Set the number of bytes in the 6 LSB bits of the control byte
10. If there are no more bytes remaining, set the 2nd MSB it of the control byte.
11. Send the frame to Vector
12. If there are byte remaining, repeat from step 2.

Encryption support). A “first time pairing” connection type will cause Vector to display a pin code on the screen

If a first time pairing response is sent:

- If Vector is not in pairing mode – was not put on his charger and the backpack button pressed twice, quickly – Vector will respond. Attempting to enter pairing mode now will cause Vector to send a *disconnect* request.
- If Vector is in pairing mode, Vector will display a pin code on the screen, and send a nonce message, triggering the next steps of the conversation.

If a reconnection is sent, the application would employ the public and secret keys, and the encryption and decryption keys from a prior pairing.

16.7. DISCONNECT

This may be sent by Vector if there is an error, and it is ending communication. For instance, if Vector enters pairing mode, it will send a disconnect.

The application may send this to request Vector to close the connection.

16.7.1 Request

The command has no parameters.

16.7.2 Response

There is no response.

16.8. FILE DOWNLOAD

This command is used to pass chunks of a file to Vector. Files are broken up into chunks, and sent.

16.8.1 Request

The parameters of the request body are:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*status*	
1	4	uint32_t	*file id*	
5	4	uint32_t	*packet number*	The chunk within the download
9	4	uint32_t	*packet total*	The total number of packets to be sent for this file download
13	2	uint12_t	*length*	The number of bytes to follow (can be 0)
	varies	uint8_t[length]	*bytes*	The bytes of this file chunk

Table 20: *Parameters for File Download request*

16.8.2 Response

There is no response [?TBD?]

16.9. LOG

This command is used to request the Vector TBD logging

16.9.1 Request

The parameters of the request body are:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*mode*	
1	2	uint16_t	*num filters*	The number of filters in the array
3	varies	filter[num filters]	*filters*	The filter names

Table 21: *Parameters for Log request*

Each filter entry has the following structure:

Offset	Size	Type	Parameter	Description
0	2	uint16_t	*filter length*	The length of the filter name; may be 0
2	*varies*	uint8_t[filter length]	*filter name*	The filter name

Table 22: *Log filter*

16.9.2 Response

The parameters for the response message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*exit code*	
1	4	uint32_t	*file id*	

Table 23: *Parameters for Log Response*

16.10. NONCE

A nonce is sent by Vector after he has accepted your key, and the application sends a response

16.10.1 Request

The parameters for the nonce request message:

Table 24: Parameters for Nonce request

Offset	Size	Type	Parameter	Description
0	24	uint8_t[24]	*toVectorNonce*	The nonce to use for sending stuff to Vector
24	24	uint8_t[24]	*toAppNonce*	The nonce for receiving stuff from Vector

16.10.2 Response

After receiving a nonce, if the application is in first-time pairing the application should send a response, with a value of 3.

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*connection tag*	This is always 3

After the response has been sent, the channel will now be encrypted. If vector likes the response, he will send a challenge message.

16.11. OTA UPDATE

This command is used to request the Vector download firmware from a given server

16.11.1 Request

The parameters of the request body are:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*length*	The length of the URL; may be 0
1	*varies*	uint8_t[length]	*URL*	The URL string

Table 25: *Parameters for OTA request*

16.11.2 Response

The parameters for the response message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*status*	See *Table 27: OTA status enumeration*
1	8	uint64_t	*current*	The number of bytes downloaded
9	8	uint64_t	*expected*	The number of bytes expected to be downloaded

Table 26: *Parameters for OTA Response*

The OTA status are:

Index	Meaning
0	idle
1	unknown
2	in progress
3	complete
4	rebooting
5	error

Table 27: *OTA status enumeration*

16.12. RESPONSE

It is not known why this message will be sent.

Table 28: Parameters for Response

Offset	Size	Type	Parameter	Description
0	1	uint16_t	*code*	0 if not cloud authorized, otherwise authorized
1	1	uint8_t	*length*	
	varies	uint8_t [length]	*bytes*	

16.13. SDK PROXY

This command is used to pass the gRPC/protobufs messages to Vector over Bluetooth LE. It effectively wraps a HTTP request/response.

16.13.1 Request

The parameters of the request body are:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*GUID length*	The number of bytes in the GUID string; may be 0
2	*varies*	uint8_t[GUID length]	*GUID*	The GUID string
	1	uint8_t	*msg length*	The number of bytes in the message id string
	varies	uint8_t[msg id length]	*msg id*	The message id string
	1	uint8_t	*path length*	The number of bytes in the URL path string
	varies	uint8_t[path length]	*path*	The URL path string
	2	uint16_t	*JSON length*	The length of the JSON
	varies	uint8_t[JSON length]	*JSON*	The JSON (string)

Table 29: *Parameters for the SDK proxy request*

16.13.2 Response

The parameters for the response message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*msg id length*	The number of bytes in the message id string; may be 0
2	*varies*	uint8_t[msg id length]	*msg id*	The message id string
	2	uint16_t	*status code*	The HTTP-style status code that the SDK may return.
	1	uint8_t	*type length*	The number of bytes in the response type string
	varies	uint8_t[type length]	*type*	The response type string
	2	uint16_t	*body length*	The length of the response body
	varies	uint8_t[body length]	*body*	The response body (string)

Table 30: *Parameters for the SDK proxy Response*

16.14. SSH

This command is used to request the Vector allow SSH. It is not known which version of the Vector support SSH, or whether it is enabled in the release.

16.14.1 Request

The parameters for the request message:

Table 31: Parameters for SSH request

Offset	Size	Type	Parameter	Description
0	2	uint16_t	*num keys*	The number of SSH authorization keys; may be 0
2	*varies*	keys[num keys]	*keys*	The array of authorization key strings (see below).

Each authorization key has the following structure:

Table 32: SSH authorization key

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*key length*	The length of the key; may be 0
1	*varies*	uint8_t[key length]	*key*	The SSH authorization key

16.14.2 Response

The response has no parameters

16.15. STATUS

This command is used to request the Vector act basic info.

16.15.1 Request

The request has no parameters

16.15.2 Response

The parameters for the response message:

***Table 33:** Parameters for Status Response*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*SSID length*	The number of bytes in the SSID string; may be 0
2	*varies*	uint8_t[SSID length]	*SSID*	The WiFi SSID (hex string)
	1	uint8_t	*WiFi state*	See *Table 34: WiFi state enumeration*
	1	uint8_t	*access point*	0 not acting as an access point, otherwise acting as an access point
	1	uint8_t	*Bluetooth LE state*	
	1	uint8_t	*Battery state*	
	1	uint8_t	*version length*	The number of bytes in the version string; may be 0 version >= 2
	varies	uint8_t [version length]	*version*	The version string; version >= 2
	1	uint8_t	*ESN length*	The number of bytes in the ESN string; may be 0 version >= 4
	varies	uint8_t[ESN length]	*ESN*	The *electronic serial number* string; version >= 4
	1	uint8_t	*OTA in progress*	0 over the air update not in progress, otherwise in process of over the air update; version >= 2
	1	uint8_t	*has owner*	0 does not have owner, otherwise has owner; version >= 3
	1	uint8_t	*cloud authorized*	0 is not cloud authorized, otherwise is cloud authorized; version >= 5

Note: a *hex string* is a series of bytes with values 0-15. Every pair of bytes must be converted to a single byte to get the characters. Even bytes are the high nibble, odd bytes are the low nibble.

The WiFi states are:

***Table 34:** WiFi state enumeration*

Index	Meaning
0	Unknown
1	Online
2	Connected
3	Disconnected

16.16. WIFI ACCESS POINT

This command is used to request that the Vector act as a WiFi access point.

16.16.1 Request

The parameters of the request body are:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*enable*	0 to disable the WiFi access point, 1 to enable it

Table 35: *Parameters for WiFi Access Point request*

16.16.2 Response

The parameters for the response message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*enabled*	0 if the WiFi access point is disabled, otherwise enabled
1	1	uint8_t	*SSID length*	The number of bytes in the SSID string; may be 0
2	*varies*	uint8_t[SSID length]	*SSID*	The WiFi SSID (hex string)
	1	uint8_t	*password length*	The number of bytes in the password string; may be 0
	varies	uint8_t [password length]	*password*	The WiFi password

Table 36: *Parameters for WiFi Access Point Response*

16.17. WIFI CONNECT

This command is used to request the Vectors connect to a given WiFi SSID. Vector will retain this WiFi for future use.

16.17.1 Request

The parameters for the request message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*SSID length*	The number of bytes in the SSID string; may be 0
1	*varies*	uint8_t[SSID length]	*SSID*	The WiFi SSID (hex string)
	1	uint8_t	*password length*	The number of bytes in the password string; may be 0
	varies	uint8_t [password length]	*password*	The WiFi password
	1	uint8_t	*timeout*	How long to given the connect attempt to succeed.
	1	uint8_t	*auth type*	The type of authentication to employ; see *Table 38: WiFi authentication types enumeration*
	1	uint8_t	*hidden*	0 the access point is not hidden; 1 it is hidden

***Table 37:** Parameters for WiFi Connect request*

The WiFi authentication types are:

Index	Meaning
0	None, open
1	WEP
2	WEP shared
3	IEEE8021X
4	WPA PSK
5	WPA2 PSK
6	WPA2 EAP

***Table 38:** WiFi authentication types enumeration*

16.17.2 Response

The parameters for the response message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*SSID length*	The length of the SSID that was deleted; may be 0
1	*varies*	uint8_t[SSID length]	*SSID*	The SSID (hex string) that was deleted
	1	uint8_t	*WiFi state*	See *Table 34: WiFi state enumeration*
	1	uint8_t	*connect result*	version >= 3

***Table 39:** Parameters for WiFi Connect command*

16.18. WIFI FORGET

This command is used to request the Vectors forget a WiFi SSID.

16.18.1 Request

The parameters for the request message:

Table 40: *Parameters for WiFi Forget request*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*delete all*	0 if Vector should delete only one SSID; otherwise Vector should delete all SSIDs
1	1	uint8_t	*SSID length*	The length of the SSID that to be deleted; may be 0
2	*varies*	uint8_t[SSID length]	*SSID*	The SSID (hex string) to be deleted

16.18.2 Response

The parameters for the response message:

Table 41: *Parameters for WiFi Forget response*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*did delete all*	0 if only one; otherwise Vector deleted all SSIDs
1	1	uint8_t	*SSID length*	The length of the SSID that was deleted; may be 0
2	*varies*	uint8_t[SSID length]	*SSID*	The SSID (hex string) that was deleted

16.19. WIFI IP ADDRESS

This command is used to request the Vectors WiFi IP address.

16.19.1 Request

The request has no parameters

16.19.2 Response

The parameters for the response message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*has IPv4*	0 if Vector doesn't have an IPv4 address; other it does
1	1	uint8_t	*has IPv6*	0 if Vector doesn't have an IPv6 address; other it does
2	4	uint8_t[4]	*IPv4 address*	Vector's IPv4 address
6	32	uint8_t[16]	*IPv6 address*	Vector's IPv6 address

Table 42: *Parameters for WiFi IP Address response*

16.20. WIFI SCAN

This command is used to request the Vectors scan for WiFi access points.

16.20.1 Request

The command has no parameters.

16.20.2 Response

The parameters for the response message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*status code*	
1	1	uint8_t	*num entries*	The number of access points in the array below
2	*varies*	AP[num entries]	*access points*	The array of access points

Table 43: *Parameters for WiFi scan response*

Each access point has the following structure:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*auth type*	The type of authentication to employ; see *Table 38: WiFi authentication types enumeration*
1	1	uint8_t	*signal strength*	The number of bars, 0..4
2	1	uint8_t	*SSID length*	The length of the SSID string
3	*varies*	uint8_t[SSID length]	*SSID*	The SSID (hex string)
	1	uint8_t	*hidden*	0 not hidden, 1 hidden; version >= 2
	1	uint8_t	*provisioned*	0 not provisioned, 1 provisioned; version>= 3

Table 44: *Parameters access point structure*

CHAPTER 9

The Cloud Services

This chapter is about the cloud

- The user's account settings and entitlements
- The robot's settings (user preferences)
- The robot's lifetime stats
- The crash uploader
- The firmware update process
- How to extract official program files

17. THE JDOC SERVER

17.1. THE USER ACCOUNT SETTINGS

Sent to the jdoc server

Schema: TBD

17.2. ROBOT SETTINGS

Sent to the jdoc server

These are an ad hoc set of locale preferences. The following are sent to the jdoc server by the mobile application, and retrieved by the robot:

***Table 45:** The robot settings.*

Key	units	Description & Notes
button_wakeword	?	
clock_24_hour	boolean	
default_location		
dist_is_metric		
eye_color		The color used for the eyes
locale		American English, UK English, Australian English, German, French, Japanese, etc.
master_volume		
temp_is_fahrenheit		
time_zone		

17.3. USER ENTITLEMENTS

Sent to the jdoc server

Schema: TBD

Appear to be the private and public keys

17.4. ROBOT LIFETIME STATS

Sent to the jdoc server

The following are held in the server, updated by the robot (I don't know if the robot has a local copy), and retrievable by the application. A JSON hash of the following:

***Table 46:** The robot lifetime stats schema*

Key	units	Description & Notes
Alive.seconds	seconds	Vectors age, since he was given preferences (a factory reset restarts this)
Stim.CumlPosDelta		Cumulative stimulation of some kind
BStat.AnimationPlayed	count	The number of animations played
BStat.BehaviorActivated	count	
BStat.AttemptedFistBump	count	The number of fist bumps (attempted)
BStat.FistBumpSuccess	count	
BStat.PettingBlissIncrease		
BStat.PettingReachedMaxBliss		
BStat.ReactedToCliff	count	
BStat.ReactedToEyeContact	count	
BStat.ReactedToMotion	count	
BStat.ReactedToSound	count	
BStat.ReactedToTriggerWord	count	
Feature.AI.DanceToTheBeat		
Feature.AI.Exploring		
Feature.AI.FistBump		
Feature.AI.GoHome		
Feature.AI.InTheAir		
Feature.AI.InteractWithFaces	count	The number of times recognized / interacted with faces
Feature.AI.Keepaway		
Feature.AI.ListeningForBeats		
Feature.AI.LowBattery		
Feature.AI.Observing		
Feature.AI.ObservingOnCharger		
Feature.AI.Onboarding		
Feature.AI.Sleeping		

Feature.AI.Petting		
Feature.AI.ReactToCliff		
Feature.AI.StuckOnEdge		
Feature.AI.UnmatchedVoiceIntent		
Feature.Voice.VC_Greeting		
FeatureType.Autonomous		
FeatureType.Failure		
FeatureType.Sleep		
FeatureType.Social		
FeatureType.Play		
FeatureType.Utility1		
Pet.ms	ms	The number of milliseconds petted?
Odom.LWheel		The left wheel odometer
'Odom.Rwheel		The right wheel odometer
Odom.Body		

18. DAS

DAS communication events, sent/received, home network stuff, etc.
seems to allow detailed reconstruction of the setup, configuration and interaction

Seems to gather mostly information from the mobile application – the name of each button pressed, screen displayed, error encountered, etc. probably gathers info about crashes, driving, cliff sensor

Might get other activation information, used for the lifetime stats

References & Resources

Note: most references appear in the margins, significant references will appear at the end of their respective chapter.

19. CREDITS

Credit and thanks to Anki, C0RE, Melanie T for access to the partition file-system, and decode keys; board shots. Fictiv for board shots. The board shots that help identify parts on the board and inter-connection on the board.

20. REFERENCE DOCUMENTATION AND RESOURCES

20.1. ANKI

Anki, "*Vector Quick Start Guide,"* 293-00036 Rev: B, 2018

Anki, Molly Jameson & Daria Jerjomina, "*Cozmo: Animation pipeline for a physical robot,"* 2017 Game Developers conference

Anki, Nathaniel Monson, Andrew Stein, Daniel Casner *Reducing Burn-in of Displayed Images*, Patent US 20372659 A1, 2017 Dec 28

Anki, Daniel Casner, Lee Crippen, Hanns Tappeiner, Anthony Armenta, Kevin Yoon, *Map Related Acoustic Filtering by a Mobile Robot,* Patent US 0212441 A1, 2019 Jul 11

Anki, Andrew Stein, *Decoding Machine-Readable Optical codes with Aesthetic Component*, Patent US 9,607,199 B2, 2017 Mar. 28

20.2. OTHER

FCC *ID 2AAIC00010 internal photos*
https://fccid.io/2AAIC00010

FCC *ID 2AAIC00011 internal photos*
https://fccid.io/2AAIC00011

Fictiv, Swetha Sriram, *Anki Vector Robot Teardown*, 2019 Aug 6

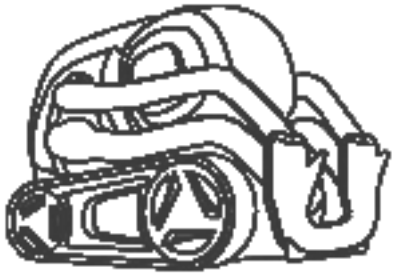
https://www.fictiv.com/blog/anki-vector-robot-teardown

Kinvert, *Anki Vector Customer Care Info Screen (CCIS)*
https://www.kinvert.com/anki-vector-customer-care-info-screen-ccis/

[This page is intentionally left blank for purposes of double-sided printing]

Appendices

- ABBREVIATIONS, ACRONYMS, & GLOSSARY. This appendix provides a gloss of terms, abbreviations, and acronyms.
- TOOL CHAIN. This appendix lists the tools known or suspected to have been used by Anki to create, and customize the Vector, and for the servers. Tools that can be used to analyze vector
- BLUETOOTH LE PROTOCOLS. This appendix provides information on the Bluetooth LE interfaces to the companion Cube, and to Anki Vector
- SERVERS. This appendix provides the servers that the Anki Vector and App contacts
- PHRASES. This appendix reproduces the phrases that the Vector keys off of.
- FEATURES. This appendix enumerates the Vector OS "features" that can be enabled and disabled.
- FILE SYSTEM. This appendix lists the key files that are baked into the system.
- PLEO. This appendix gives a brief overview of the Pleo animatronic dinosaur.



[This page is intentionally left blank for purposes of double-sided printing]

APPENDIX A

Abbreviations, Acronyms, Glossary

Abbreviation / Acronym	Phrase
ADC	analog to digital converter
AG	animation group
AVS	Alexa Voice Service
BIN	binary file
CCIS	customer care information screen
CLAD	C-like abstract data structures
CRC	cyclic redundancy check
DAS	data analytics service ?
DFU	device firmware upgrade
EEPROM	electrical-erasable programmable read-only memory
EMR	*unknown*
ESD	electro-static discharge
ESN	electronic serial number
FBS	flat buffers
GPIO	general purpose IO
I2C	inter-IC communication
IMU	inertial measurement unit
IR	infrared
JTAG	Joint Test Action Group
LCD	liquid crystal display
LED	light emitting diode
LUKS	linux unified key setup
MCU	microcontroller
MEMS	micro-electromechanical systems
MISO	master-in, slave-out
MOSI	master-out, slave-in
MPU	microprocessor
MSRP	manufacturer's suggest retail price

Table 47: *Common acronyms and abbreviations*

OLED	organic light-emitting diode display
OTA	over the air updates
PCB	printed circuit board
PCBA	printed circuit board assembly (PCB with the components attached)
PMIC	power management IC
RPM	resource power management
RRT	rapidly-expanding random tree
SCLK	(I2C) serial clock
SDA	(I2C) serial data
SDK	software development kit
SLAM	simultaneous localization and mapping
SPI	serial-peripheral interface
SSH	secure shell
SSID	service set identifier (the name of the Wifi network)
STM32	A microcontroller family from ST Microelectronics
SWD	single wire debug
TTS	text to speech
UART	universal asynchronous receiver/transmitter
USB	universal serial bus
UUID	universally unique identifier
vic	short Victor (Vector's working name)

Phrase	Description
A*	A path finding algorithm
attitude	orientation
bootloader	A piece of software used to load and launch the application firmware.
C-like abstract data structure	Anki's phrase for when information is packed into fields and values with a defined binary format, and interpretation. .(Protobufs are often used for the same purpose.)
capacitive touch	
characteristic (Bluetooth LE)	A key (or slot) that holds a value in the services key-value table. A characteristic is uniquely identified by its UUID.
control	motors and forces to move where and how it is told to. (smooth arcs)
D*-lite	A path-finding algorithm
flash	A type of persistent (non-volatile) storage media.
guidance	desired path
navigation	knowing where it is in the map
nonce	An initially random number, incremented after each use .

Table 48: *Glossary of common terms and phrases*

pose	position and orientation of an object relative to a coordinate system
power source	Where the electric energy comes from.
path planning	smooth arcs and line segments
rapidly-expanding random tree	A path-finding algorithm
simultaneous localization and mapping	A vision based technique for building a map of the immediate world for purposes of positioning oneself within it, and detecting relative movements.
service (Bluetooth LE)	A key-value table, grouped together for a common purpose. A service is uniquely identified by its UUID.
universally unique identifier (UUID)	A 128bits number that is unique.

APPENDIX B

Tool chain

This appendix tries to capture the tools that Anki is known or suspected to have used for the Anki Vector and its cloud server.

Note: Several of these the licenses requiring Anki to post their versions of the GPL tools, and their modification, Anki never did. Qualcomm may have; as the license requirement only to those their customer, they may have provided the changes to them.

***Table 49:** Tools used by Anki*

Tool	Description
Acapela	Vector uses Acapela's text to speech synthesizer, and the Ben voice. https://www.acapela-group.com/
Advanced Linux Sound Architecture (alsa)	The audio system https://www.alsa-project.org
Amazon Alexa	A set of software tools that allows Vector to integrate Alexa voice commands, probably in the AMAZONLITE distribution https://developer.amazon.com/alexa-voice-service/sdk
Amazon Web services	used on the server https://aws.amazon.com/
android boot-loader	Vector uses the Android Boot-loader;
ARM NN	ARM's neural network support https://github.com/ARM-software/armnn
AudioKinetic Wwise[18]	Used to craft the sounds https://www.audiokinetic.com/products/wwise/
clang	A C/C++ compiler, part of the LLVM family https://clang.llvm.org
bluez5	Bluetooth LE support http://www.bluez.org/
busybox	The shell on the Anki Vector linux https://busybox.net
chromium update	?
civetweb	The embedded webserver that allows Mobile apps and the python SDK to communicate with Vector. https://github.com/civetweb/civetweb
connman	Connection manager for WiFi https://01.org/connman
Google FlatBuffers	Google FlatBuffers is used to encode the animation data structures https://github.com/google/flatbuffers

[18] https://blog.audiokinetic.com/interactive-audio-brings-cozmo-to-life/?hsFormKey=227ccf4a650a1cffd6562c16d655a0ef

GNU C Compiler (gcc)	GCC version 4.9.3 was used to compile the kernel
golang	Go is used on the server applications, and (reported) some of Vectors internal software.
Google RPC (gRPC)	A "remote procedure call" standard, that allows mobile apps and the python SDK to communicate with Vector. https://grpc.io/docs/quickstart/cpp/
hdr-histogram	Unknown use https://github.com/HdrHistogram/HdrHistogram
libchromatix	Qualcomm camera support
libsodium	Cryptography library suitable for the small packet size in Bluetooth LE connections. Used to encrypt the mobile applications Bluetooth LE connection with Vector. https://github.com/jedisct1/libsodium
linux, yocto[19]	The family of linux distribution used for the Anki Vector (v3.18.66)
linux	on the server
linux unified key storage (LUKS)	
Maya	A character animation tool set, used to design the look and movements of Cozmo and Vector. The tool emitted the animation scripts.
mpg123	A MPEG audio decoder and player. This is needed by Alexa; other uses are unknown. https://www.mpg123.de/index.shtml
ogg vorbis	Audio codec https://xiph.org/vorbis
open CV	Used for the first-level image processing – to locate faces, hands, and possibly accessory symbols. https://opencv.org/
openssl	used to validate firmware update signature https://www.openssl.org
opkg	Package manager, from yocto https://git.yoctoproject.org/cgit/cgit.cgi/opkg/
Opus codec	Audio codec; may be used to encode speech sent to servers http://opus-codec.org/
perl	A programming language, on Victor https://www.perl.org
protobuf	Used to describe the format/encoding of data sent over gRPC to and from Vector. This is used by mobile and python SDK, as well as on the server. https://developers.google.com/protocol-buffers
Pryon	The recognition for the Alexa keyword at least the file system includes the same model as distributed in AMAZONLITE https://www.pryon.com/company/
python	A programming language and framework, used with desktop tools to communicate with Vector. Vector has python installed. Probably used on the server as well. https://www.python.org

[19] https://www.designnews.com/electronics-test/lessons-after-failure-anki-robotics/140103493460822

Qualcomm TBD	Qualcomm's device drivers, and other kit appears to be used.
Segger ICD	A high-end ARM compatible in-circuit debugging probe. Rumoured to have been used by Anki engineers, probably with the STM32F030 https://www.segger.com/products/debug-probes/j-link/
Sensory	Includes recognition for "Hey Vector" and Alexa wake word by Sensory, Inc. https://www.sensory.com/products/technologies/trulyhandsfree/ https://en.wikipedia.org/wiki/Sensory,_Inc.
SQLite	This is needed by Alexa; other uses are unknown https://www.sqlite.org/index.html
systemd	Used by Vector to launch the internal services https://www.freedesktop.org/software/systemd/
tensor flow lite (TFLite)	Probably used to recognize the object marker symbols, and maybe hands https://www.tensorflow.org/lite/microcontrollers/get_started

Other tools, useful for analyzing and patching Vector:

Table 50: *Tools that can be used to analyze and patch Vector*

Toool	**Description**
Segger ICD	An education version of the J-Link, suitable for the STM32F030, can be found on ebay for <$60 https://www.segger.com/products/debug-probes/j-link/
ST-Link (v3)	Suitable for extracting the STM32F030 and installing patched firmware; $35 https://www.st.com/en/development-tools/stlink-v3set.html
TI BLE sniffer	$50 http://www.ti.com/tool/CC2540EMK-USB https://www.ti.com/tool/PACKET-SNIFFER
Wireshark	To decode what is said to the servers https://support.citrix.com/article/CTX116557

APPENDIX C

Bluetooth LE Services & Characteristics

This Appendix describes the configuration of the Bluetooth LE services – and the data access they provide – for the accessory cube and for Vector.

21. CUBE SERVICES

times and other feature parameters:

Service	UUID[20]	Description & Notes
Device Info Service[21]	$180A_{16}$	Provides device and unit specific info – it's manufacturer, model number, hardware and firmware versions
Generic Access Profile[22]	1800_{16}	The device name, and preferred connection parameters
Generic Attribute Transport[23]	1801_{16}	Provides access to the services.
Cube's Service	C6F6C70F-D219-598B-FB4C-308E1F22F830$_{16}$	Service custom to the cube, reporting battery, accelerometer and date of manufacture

***Table 51:** The Bluetooth LE services*

Note: It appears that there isn't a battery service on the Cube. When in over-the-air update mode, there may be other services present (i.e. by a bootloader)

Element	Value
Device Name (Default)	"Vector Cube"
Firmware Revision	"v_5.0.4"
Manufacturer Name	"Anki"
Model Number	"Production"
Software Revision	"2.0.0"

***Table 52:** The Cube's Device info settings*

[20] All values are a little endian, per the Bluetooth 4.0 GATT specification

[21] http://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.device_information.xml

[22] http://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.generic_access.xml

[23] http://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.generic_attribute.xml

21.1. CUBE'S ACCELEROMETER SERVICE

Values are little-endian, except where otherwise stated.

UUID	Access	Size	Notes
0EA75290-6759-A58D-7948-598C4E02D94A$_{16}$	*Write*	unknown	
450AA175-8D85-16A6-9148-D50E2EB7B79E$_{16}$	Read	The date and time of manufacture (?)	
		char[]	*A date and time string*
43EF14AF-5FB1-7B81-3647-2A9477824CAB$_{16}$	Read, Notify, Indicate	Reads the battery and acceleromter	
		uint16_t	*battery ADC value*
		uint16_t	*accelerometer X ADC value #1*
		uint16_t	*accelerometer Y ADC value #1*
		uint16_t	*accelerometer Z ADC value #1*
		uint16_t	*accelerometer X ADC value #2*
		uint16_t	*accelerometer Y ADC value #2*
		uint16_t	*accelerometer Z ADC value #2*
		uint16_t	*accelerometer X ADC value #3*
		uint16_t	*accelerometer Y ADC value #3*
		uint16_t	*accelerometer Z ADC value #3*
		uint16_t	
9590BA9C-5140-92B5-1844-5F9D681557A4$_{16}$	Write		Unknown

***Table 53:** Cube's accelerometer service characteristics*

Presumably some of these will cause the Cube to go into over the air update (OTAU) mode, allowing its firmware to be updated.

Others turn the RGB on to an RGB color, possibly duty cycle and pulsing duty cycle

22. VECTOR SERVICES SERVICE

times and other feature parameters:

Service	UUID[24]	Description & Notes
Generic Access Profile	1800_{16}	The device name, and preferred connection parameters
Generic Attribute Transport	1801_{16}	Provides access to the services.
Vectors Serial Service	$FEE3_{16}$	The service with which we can talk to Vector.

***Table 54:** Vector's Bluetooth LE services*

It appears that there isn't a battery service on the Vector.

Element	Value
Device Name (Default)	"Vector" followed by his serial number

***Table 55:** The Vector's Device info settings*

[24] All values are a little endian, per the Bluetooth 4.0 GATT specification

22.1. VECTORS SERIAL SERVICE

UUID	Access	Format	Notes
30619F2D-0F54-41BD-A65A-7588D8C85B45$_{16}$	Read, Notify,Indicate		
7D2A4BDA-D29B-4152-B725-2491478C5CD7$_{16}$	write		

Table 56: *Vector's serial service characteristics*

APPENDIX D

Servers & Data Schema

This Appendix describes the servers that Vector contacts[25]

Server	Description & Notes
chipper.api.anki.com:443	The speech recognition engine lives here
conncheck.global.anki-services.com/ok	Used to check to see if it can connect to Anki
jdocs.api.anki.com:443	Storage of some sort of data. Name, faces, prefs?
token.api.anki.com:443	Used to get the API certificate.[26]
https://anki.sp.backtrace.io:6098	Vector posts crashes to this server
https://sqs.us-west-2.amazonaws.com/792379844846/DasProd-dasprodSqs-1845FTIME3RHN	This is used to synchronize with data analytics services.
https://ota.global.anki-services.com/vic/prod/	Where Vector checks for updates
https://ota.global.anki-dev-services.com/vic/rc/lo8awreh23498sf/	For the Developer branch
amazon.com/code	Where does the visual go?

Table 57: *The servers that Vector contacts.*

The servers that the mobile app contacts:

TBD

[25] Todo: sync up with info at: https://github.com/anki-community/vector-archive
[26] XYZ had a write up, reference that.

APPENDIX E

Phrases and their Intent

This Appendix maps the published phrases that Vector responds to and their intent:

Table 58: *The "Hey Vector" phrases*

Intent	Phrase
movement_backward	Back up
imperative_scold	Bad robot
	Be quiet
global_stop	Cancel the timer
	Change/set your eye color to [blue, green, lime, orange, purple, sapphire, teal, yellow].
check_timer	Check the timer
imperative_come	Come here
imperative_dance	Dance.
play_popawheelie	Do a wheelstand
imperative_fetchcube	Fetch your cube
imperative_findcube	Find your cube
play_fistbump	Fist Bump
play_fistbump	Give me a Fist Bump
movement_backward	Go backward
explore_start	Go explore
movement_forward	Go forward.
movement_turnleft	Go left
movement_turnright	Go right
	Go to sleep
	Go to your charger
	Good afternoon
greeting_goodbye	Goodbye
	Good evening
	Good night

greeting_goodmorning	Good morning
imperative_praise	Good robot
seasonal_happyholidays	Happy Holidays
seasonal_happynewyear	Happy New Year
greeting_hello	Hello
	He’s behind you
character_age	How old are you
imperative_abuse	I hate you.
knowledge_question	I have a question …
imperative_love	I love you.
imperative_apology	I’m sorry.
play_blackjack	Let’s play Blackjack
	Listen to music
imperative_lookatme	Look at me
	Look behind you
	My name is [Your Name]
imperative_negative	No
play_anygame	Play a game
play_anytrick	Play a trick
play_blackjack	Play Blackjack
play_pickupcube	Pick up your cube.
play_popawheelie	Pop a wheelie.
play_rollcube	Roll your Cube
	Run
set_timer	Set a timer for [length of time]
explore_start	Start Exploring
	Stop Exploring
global_stop	Stop the timer
take_a_photo	Take a picture of [me/us]
take_a_photo	Take a picture
take_a_photo	Take a selfie
movement_turnaround	Turn around
movement_turnleft	Turn left
movement_turnright	Turn right
imperative_volumelevel	Volume [number].
imperative_volumedown	Volume down
imperative_volumeup	Volume up.

	Volume maximum
names_ask	What's my name?
weather_response	What's the weather in [City Name]?
weather_response	What's the weather report?
show_clock	What time is it?
imperative_affirmative	Yes

Note: Vector's NLP server doesn't recognize "home" ..

Questions

Table 59: *The Vector questions phrases*

Subject	**Example Phrase**
Current conversion	What's 1000 Yen in US Dollars?
Flight status	What is the status of American Airlines Flight 100?
Equation solver	What is the square root of 144?
General knowledge	What is the tallest building?
places	What is the distance between London and New York?
People	Who is Jarvis?
Nutrition	How many calories are in an avocado?
Sports	Who won the World Series?
Stock market	How is the stock market?
Time zone	What time is it in Hong Kong?
Unit conversion	How fast is a knot?
Word definition	What is the definition of Artificial Intelligence?

Some of them are internal strings, some are hardcoded values

APPENDIX G

File system

This Appendix describes files systems

As the Vector uses the Android bootloader, it reuses – or at least reserves – many of the Android partitions[27] and file systems. Many are probably not used. Quotes are from Android documentation.

The file system table tells use where they are stored in the partitions, and whether they are non volatile.

Mount point	Partition name	Description & Notes
/	BOOT_A	The primary linux kernel and initramfs
/data[28]	USERDATA	The data created for the specific robot (and user) that customizes it. A factory reset wipes out this user data. This portion of the file system is encrypted using "Linux Unified Key Setup" (LUKS).
/firmware	MODEM	The firmware for the WiFi/Bluetooth radio
/factory	OEM	Customizations, such as bootloader property values. … Or the factory recovery?
/persist	PERSIST	Device specific "data which shouldn't be changed after the device is shipped, e.g. DRM related files, sensor reg file (sns.reg) and calibration data of chips; wifi, bluetooth, camera etc."
/media/ram /run /var/volatile /dev/sm		Internal temporary file systems; holds temporary files, interprocess communication

***Table 60:** The file system mount table*

The partition table found on the Vector:

Partition name	Size	Description & Notes
ABOOT	1 MB	The application bootloader, which may load the kernel, recovery, or fastboot.
ABOOTBAK	1 MB	The application bootloader, which may load the kernel, recovery, or fastboot.
BOOT_A	32 MB	The primary linux kernel and initramfs.
BOOT_B	32 MB	The backup linux kernel and initramfs.
CONFIG	512 KB	Configuration of TBD.
DDR	32 KB	Configuration of the DDR RAM.
DEVINFO	1 MB	"device information including: is_unlocked (aboot), is_tampered, is_verified, charger_screen_enabled, display_panel, bootloader_version, radio_version etc.

***Table 61:** The partition table*

[27] https://forum.xda-developers.com/android/general/info-android-device-partitions-basic-t3586565
[28] This is mounted by "mount-data.service' The file has a lot of information on how it unbricks

		Contents of this partition are displayed by "fastboot oem device-info" command in human readable format. Before loading boot.img or recovery.img, bootloader verifies the locked state from this partition."
EMR	16 MB	????
FSC	1KB	"Modem FileSystem Cookies"
FSG	1.5 MB	Golden backup copy of MODEMST1, used to restore it in the event of error
KEYSTORE	512 KB	"Related to [USERDATA] Full Disk Encryption (FDE)"
MISC	1MB	"a tiny partition used by recovery to communicate with bootloader store away some information about what it's doing in case the device is restarted while the OTA package is being applied. It is a boot mode selector used to pass data among various stages of the boot chain (boot into recovery mode, fastboot etc.). e.g. if it is empty (all zero), system boots normally. If it contains recovery mode selector, system boots into recovery mode."
MODEM	64 MB	Binary "blob" for the WiFi/Bluetooth radio firmware
MODEMST1	1.5MB	Binary "blob" for the WiFi/Bluetooth radio firmware
MODEMST1	1.5MB	Binary "blob" for the WiFi/Bluetooth radio firmware
OEM	16MB	Customizations, such as bootloader property values.
PAD	1MB	"related to OEM"
PERSIST	64MB	Device specific "data which shouldn't be changed after the device is shipped, e.g. DRM related files, sensor reg file (sns.reg) and calibration data of chips; wifi, bluetooth, camera etc."
RECOVERY	32 MB	An alternate partition holding systems applications and libraries that let the application boot into a mode that can download a new system. Often used to wipe out the updates.
RECOVERYFS	640 MB	An alternate partition holding systems applications and libraries that let the application boot into a mode that can download a new system. Often used to wipe out the updates.
RPM	512KB	The primary for resource power management[?]
RPMBAK	512KB	The backup for resource power management[?]
SBL1	512KB	Secondary bootloader. Responsible for loading ABOOT; may also have an "Emergency" download mode.
SBL1BAK	512KB	Secondary bootloader. Responsible for loading ABOOT; may also have an "Emergency" download mode.
SEC	16KB	The secure boot fuse settings, OEM settings, signed-bootloader stuff
SSD	8KB	"Secure software download" for secure storage, encrypted RSA keys, etc
SYSTEM_A	896MB	The (primary) systems applications and libraries with application specific code.
SYSTEM_B	896MB	The bakcup systems applications and libraries with application specific code.
SWITCHBOARD	16 MB	????
TZ	768KB	The TrustZone of key-value pairs, encrypted by the hardware and other keys
TZBAK	768KB	The TrustZone of key-value pairs, encrypted by the hardware and other keys
USERDATA	768MB	The data created for the specific robot (and user) that customizes it. A factory reset wipes out this user data. This partition is encrypted using "Linux Unified Key Setup" (LUKS).

The files employed in the Vector binaries:

Table 62: *Files*

File	Description
/anki/etc/version	
/data/data/com.anki.victor	
/data/data/com.anki.victor/cache/crashDumps	
/data/etc/localtime	The time zone
/data/misc/bluetooth/abtd.socket	The IPC socket interface to Bluetooth LE
/data/unbrick	
/dev/block/bootdevice/by-name/emr	
*/dev/socket/_anim_robot_server_*	The IPC socket with Vector's animation controller
*/dev/socket/_engine_gateway_server_*	The IPC socket interface to Vector's Gateway [TBD] server
*/dev/socket/_engine_gateway_proto_server_*	The IPC socket interface to Vector's Gateway [TBD] server
*/dev/socket/_engine_switch_server_*	The IPC socket interface to Vector's Switchbox [TBD] server
/factory/cloud/something.pem	
/proc/sys/kernel/random/boot_id	A random identifier, created each boot
/sys/devices/system/cpu/possible[29] */sys/devices/system/cpu/present*	The number of CPUs and whether they can be used.
/data/data/com.anki.victor/cache/crashDumps	Parameters and values specific to the ST LIS2DH accelerometer
/run/anki-crash-log	
/run/das_allow_upload	
/run/fake-hwclock-cmd[30]	Sets the fake time to the time file (Vector doesn't have a clock)
/run/fault_code	
/tmp/data_cleared	
/tmp/vision/neural_nets	

Key named device files employed in Vector binaries:

Table 63: *Named /dev device files*

File	Description
/dev/fb0	The display framebuffer
/dev/spidev0.0	The SPI channel to communicate with the IMU
/dev/spidev1.0	The SPI channel to communicate with the LCD
/dev/ttyHS0	Console log? (but then, where is the connection to the base-board?)
/dev/ttyHSL0	
/sys/class/gpio/gpio83	Used to control the camera power

[29] https://www.kernel.org/doc/Documentation/ABI/testing/sysfs-devices-system-cpu
[30] https://manpages.debian.org/jessie/fake-hwclock/fake-hwclock.8.en.html

APPENDIX H

Pleo

The Pleo, sold in 2007 –a decade prior to Vector – has many similarities. The Pleo was a software skinned animatronic baby dinosaur created by Caleb Chung, John Sosuka and their team at Ugobe. Ugobe went bankrupt in 2009, and the rights were bought by Innvo Labs which introduced a second generation in 2010. This appendix is mostly adapted from the Wikipedia article and reference manual.

Sensing for interacting with a person

- Two microphones, could do beat detection allowing Pleo to dance to music. The second generation (2010) could localize the sound and turn towards the source.
- 12 touch sensors (head, chin, shoulders, back, feet) to detect when petted,

Environmental sensors

- Camera-based vision system (for light detection and navigation). The first generation treated the image as gray-scale, the second generation could recognize colors and patterns.
- Four ground foot sensors to detect the ground. The second generation could prevent falling by detecting drop-offs
- Fourteen force-feedback sensors, one per joint
- Orientation tilt sensor for body position
- Infrared mouth sensor for object detection into mouth, in the first generation. The second generation could sense accessories with an RFID system.
- Infrared detection of objects
- Two-way infrared communication with other Pleos
- The second generation include a temperature sensor

Annuciators and Actuators

- 2 speakers, to give it sounds
- 14 motors
- Steel wires to move the neck and tail (these tended to break in the first generation)

The processing

- Atmel ARM7 microprocessor was the main processor.
- An NXP ARM7 processor handle the camera system, audio input
- Low-level motor control was handled by four 8-bit processors

A developers kit – originally intended to be released at the same time as the first Pleo – was released ~2010. The design included a virtual machine intended to allow "for user programming of new behaviors."[31]

22.2. SALES

Pleo's original MSRP was $350, "the wholesale cost of Pleo was $195, and the cost to manufacture each one was $140" sold ~100,000 units, ~$20 million in sales[32]

The second generation (Pleo Reborn) had an MSRP of $469

22.3. RESOURCES

Wikipedia article. https://en.wikipedia.org/wiki/Pleo

iFixit's teardown. https://www.ifixit.com/Teardown/Pleo+Teardown/597

Ugobe, *Pleo Monitor,* Rev 1.1, 2008 Aug 18

Ugobe, *Pleo Programming Guide,* Rev 2, 2008 Aug 15

[31] https://news.ycombinator.com/item?id=17755596
[32] https://www.idahostatesman.com/news/business/article59599691.html
https://www.robotshop.com/community/blog/show/the-rise-and-fall-of-pleo-a-fairwell-lecture-by-john-sosoka-former-cto-of-ugobeJohn Sosoka