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.

drawing by Steph 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

ANKI VECTOR I

A LOVE LETTER TO THE LITTLE DUDE I

PREFACE 1

1.1. CUSTOMIZATION AND PATCHING 1
2. ORGANIZATION OF THIS DOCUMENT 1

CHAPTER 2 3

OVERVIEW OF VECTOR 3

3. OVERVIEW 3
3.1. FEATURES 3
4. COZMO 5

PART I 7

ELECTRONICS DESIGN 7

CHAPTER 3 9

ELECTRONICS DESIGN DESCRIPTION 9

5. DESIGN OVERVIEW 9
5.1. POWER SOURCE AND DISTRIBUTION TREE 12
6. THE BACKPACK BOARD 13
6.1. BACKPACK CONNECTION 13
6.2. OPERATION 13
7. THE BASE-BOARD 15
7.1. POWER MANAGEMENT 16
7.2. ELECTRO-STATIC DISCHARGE (ESD) PROTECTION 19
7.3. STM32F030 MICROCONTROLLER 19
7.4. SENSING 20
7.5. MOTOR DRIVER AND CONTROL 21
7.6. COMMUNICATION 21
8. THE HEAD-BOARD (THE MAIN PROCESSOR BOARD) 22
8.1. SPEAKERS 22
8.2. CAMERA 23
8.3. THE LCD 23
8.4. TRIM, CALIBRATION SERIAL NUMBERS AND KEYS 23

8.5. MANUFACTURING TEST CONNECTOR/INTERFACE 23
9. RESOURCES 24

CHAPTER 4 25

ACCESSORY ELECTRONICS DESIGN DESCRIPTION 25

10. CHARGING STATION 25
11. CUBE 26
11.1. OVER THE AIR FIELD UPDATES 27
11.2. RESOURCES 27

PART II 29

BASIC OPERATION 29

CHAPTER 5 31

ARCHITECTURE 31

12. OVERVIEW OF VECTOR'S COMMUNICATION INFRASTRUCTURE 31
12.1. APPLICATION SERVICES ARCHITECTURE 32
13. ROBOT DATA 33
13.1. EMR 33
14. REFERENCES & RESOURCES 33

CHAPTER 6 34

POWER MANAGEMENT 34

15. STARTUP 34
15.1. REGULAR SYSTEM BOOT 35
15.2. ABNORMAL SYSTEM BOOT 36
16. SHUTDOWN 36
16.1. TURNING VECTOR OFF (INTENTIONALLY) 36
16.2. UNINTENTIONALLY 36
16.3. GOING INTO AN OFF STATE 36
17. POWER MANAGEMENT 37
17.1. BATTERY MANAGEMENT 37
17.2. RESPONSES, SHEDDING LOAD / POWER SAVING EFFORTS 38
18. RESOURCES 39

CHAPTER 7 40

AUDIO 40

19. AUDIO INPUT 40
19.1. THE MICROPHONES 41
19.2. SPATIAL AUDIO PROCESSING 41
19.3. NOISE REDUCTION 42
19.4. VOICE ACTIVITY DETECTOR AND WAKE WORD 42
19.5. CLOUD SPEECH RECOGNITION 43
19.6. CONNECTIONS WITH VIC-GATEWAY 43

PART III 45

COMMUNICATION 45

CHAPTER 9 47

COMMUNICATION 47

20. OVERVIEW OF VECTOR'S COMMUNICATION INFRASTRUCTURE 47
21. INTERNAL COMMUNICATION WITH PERIPHERALS 47
21.1. COMMUNICATION WITH THE BASE-BOARD 48
21.2. SERIAL BOOT CONSOLE 48
21.3. USB 48
22. BLUETOOTH LE 49
23. COMMUNICATING WITH MOBILE APP AND SDK 50
23.1. CERTIFICATE BASED AUTHENTICATION 50

CHAPTER 10 51

BLUETOOTH LE COMMUNICATION PROTOCOL 51

24. COMMUNICATION PROTOCOL OVERVIEW 51
24.1. SETTING UP THE COMMUNICATION CHANNEL 53
24.2. FRAGMENTATION AND REASSEMBLY 54
24.3. ENCRYPTION SUPPORT 55
24.4. THE RTS LAYER 56
24.5. FETCHING A LOG 57
25. MESSAGE FORMATS 58
25.1. APPLICATION CONNECTION ID 59
25.2. CANCEL PAIRING 60
25.3. CHALLENGE 61
25.4. CHALLENGE SUCCESS 62
25.5. CLOUD SESSION 63
25.6. CONNECT 65
25.7. DISCONNECT 66

25.8. File Download 67
25.9. Log 68
25.10. Nonce 69
25.11. OTA Update 70
25.12. Response 71
25.13. SDK Proxy 72
25.14. SSH 73
25.15. Status 74
25.16. WiFi Access Point 75
25.17. WiFi Connect 76
25.18. WiFi Forget 77
25.19. WiFi IP address 78
25.20. WiFi Scan 79

CHAPTER 12 80

THE CLOUD SERVICES 80

26. Configuration 80
27. JDocs Server 80
27.1. JDocs Interaction 81
27.2. Delete document 81
27.3. Echo test 81
27.4. Read documents 81
27.5. Read document Item 82
27.6. Write document 82
28. Natural Language Processing 82

PART IV 83

MAINTENANCE 83

CHAPTER 15 85

SETTINGS, PREFERENCES, STATISTICS 85

29. The architecture 85
30. WiFi Configuration 85
31. The Owner Account information 86
32. Preferences & Robot settings 87
33. Owner Entitlements 87
34. Robot Lifetime Stats 87

CHAPTER 16 90

THE SOFTWARE UPDATE PROCESS 90

35. THE ARCHITECTURE 90
36. THE UPDATE FILE 90
36.1. MANIFEST.INI 91
36.2. HOW TO DECRYPT THE OTA UPDATE ARCHIVE FILES 92
37. THE UPDATE PROCESS 92
37.1. STATUS DIRECTORY 92
37.2. PROCESS 93
38. RESOURCES 94

REFERENCES & RESOURCES 95

39. CREDITS 95
40. REFERENCE DOCUMENTATION AND RESOURCES 95
40.1. ANKI 95
40.2. OTHER 95
40.3. QUALCOMM 95

APPENDICES 97

APPENDIX A 99

ABBREVIATIONS, ACRONYMS, GLOSSARY 99

APPENDIX B 102

TOOL CHAIN 102

APPENDIX C 105

FAULT AND STATUS CODES 105

APPENDIX D 107

FILE SYSTEM 107

APPENDIX E 110

BLUETOOTH LE SERVICES & CHARACTERISTICS 110

41. CUBE SERVICES 110
41.1. CUBE'S ACCELEROMETER SERVICE 111

42. VECTOR SERVICES SERVICE 111
42.1. VECTOR'S SERIAL SERVICE 112

APPENDIX F 113

SERVERS & DATA SCHEMA 113

APPENDIX H 114

PHRASES AND THEIR INTENT 114

APPENDIX J 117

PLEO 117

42.2. SALES 118
42.3. RESOURCES 118

FIGURE 1: VECTOR'S MAIN FEATURES 3
FIGURE 2: VECTOR'S MAIN ELEMENTS 9
FIGURE 3: CIRCUIT BOARD TOPOLOGY 11
FIGURE 4: VECTOR'S MAIN MICROCONTROLLER CIRCUIT BOARDS 11
FIGURE 5: POWER DISTRIBUTION 12
FIGURE 6: BACKPACK BOARD BLOCK DIAGRAM 13
FIGURE 7: BASE-BOARD BLOCK DIAGRAM 15
FIGURE 8: A REPRESENTATIVE BATTERY CONNECT SWITCH 17
FIGURE 9: A REPRESENTATIVE PFET BASED REVERSED POLARITY PROTECTION 18
FIGURE 10: CHARGING PROFILE (ADAPTED FROM TEXAS INSTRUMENTS) 18
FIGURE 11: MOTOR DRIVER H-BRIDGE 21
FIGURE 12: HEAD-BOARD BLOCK DIAGRAM 22
FIGURE 13: CHARGING STATION MAIN FEATURES 25
FIGURE 14: CHARGING STATION BLOCK DIAGRAM 25
FIGURE 15: CUBE'S MAIN FEATURES 26
FIGURE 16: BLOCK DIAGRAM OF THE CUBE'S ELECTRONICS 26
FIGURE 17: THE OVERALL FUNCTIONAL BLOCK DIAGRAM 31
FIGURE 18: THE OVERALL COMMUNICATION INFRASTRUCTURE 32
FIGURE 19: THE BATTERY LEVEL CLASSIFICATION TREE 37
FIGURE 20: THE BATTERY THRESHOLDS 38
FIGURE 21: THE RESPONSE TO BATTERY LEVEL 39
FIGURE 22: THE AUDIO INPUT FUNCTIONAL BLOCK DIAGRAM 40
FIGURE 23: THE AUDIO INPUT ARCHITECTURE 41
FIGURE 24: THE SPATIAL AUDIO PROCESSING 41
FIGURE 25: THE AUDIO NOISE REDUCTION 42
FIGURE 26: THE SPEECH RECOGNITION 43
FIGURE 27: THE OVERALL COMMUNICATION INFRASTRUCTURE 47
FIGURE 28: THE BLUETOOTH LE STACK 49
FIGURE 29: OVERVIEW OF ENCRYPTION AND FRAGMENTATION STACK 52
FIGURE 30: SEQUENCE FOR INITIATING COMMUNICATION WITH VECTOR 53

FIGURE 31: THE FORMAT OF AN RTS FRAME 56
FIGURE 32: SEQUENCE FOR INITIATING COMMUNICATION WITH VECTOR 57
FIGURE 33: THE ARCHITECTURE FOR STORING PREFERENCES, ACCOUNT INFO, ENTITLEMENTS, AND TRACKING STATS 85
FIGURE 34: THE ARCHITECTURE FOR UPDATING VECTOR'S SOFTWARE 90

TABLE 1: VECTOR'S MAIN ELEMENTS 10
TABLE 2: VECTOR'S CIRCUIT BOARDS 11
TABLE 3: BACKPACK BOARD FUNCTIONAL ELEMENTS 13
TABLE 4: THE BASE-BOARD FUNCTIONAL ELEMENTS 15
TABLE 5: THE HEAD-BOARDS FUNCTIONAL ELEMENTS 22
TABLE 6: THE CUBE'S ELECTRONIC DESIGN ELEMENTS 26
TABLE 7: VECTOR PROCESSES 32
TABLE 8: VECTOR SHUTDOWN CODES 36
TABLE 9: BATTERYLEVEL CODES AS THEY APPLY TO VECTOR 37
TABLE 10: BATTERY STATISTICS, POSSIBLY POSTED TO DAS 38
TABLE 11: ELEMENTS OF THE BLUETOOTH LE STACK 49
TABLE 12: ELEMENTS OF A VECTOR CERTIFICATE 50
TABLE 13: PARAMETERS FOR HANDSHAKE MESSAGE 53
TABLE 14: THE ENCRYPTION VARIABLES 55
TABLE 15: SUMMARY OF THE COMMANDS 58
TABLE 16: PARAMETERS FOR APPLICATION CONNECTION ID REQUEST 59
TABLE 17: PARAMETERS FOR CHALLENGE REQUEST 61
TABLE 18: PARAMETERS FOR CHALLENGE RESPONSE 61
TABLE 19: PARAMETERS FOR CLOUD SESSION REQUEST 63
TABLE 20: PARAMETERS FOR CLOUD SESSION RESPONSE 63
TABLE 21: CLOUD STATUS ENUMERATION 63
TABLE 22: PARAMETERS FOR CONNECTION REQUEST 65
TABLE 23: PARAMETERS FOR CONNECTION RESPONSE 65
TABLE 24: CONNECTION TYPES ENUMERATION 65
TABLE 25: PARAMETERS FOR FILE DOWNLOAD REQUEST 67
TABLE 26: PARAMETERS FOR LOG REQUEST 68
TABLE 27: LOG FILTER 68
TABLE 28: PARAMETERS FOR LOG RESPONSE 68
TABLE 29: PARAMETERS FOR NONCE REQUEST 69
TABLE 30: PARAMETERS FOR NONCE RESPONSE 69
TABLE 31: PARAMETERS FOR OTA REQUEST 70
TABLE 32: PARAMETERS FOR OTA RESPONSE 70
TABLE 33: OTA STATUS ENUMERATION 70
TABLE 34: PARAMETERS FOR RESPONSE 71
TABLE 35: PARAMETERS FOR THE SDK PROXY REQUEST 72
TABLE 36: PARAMETERS FOR THE SDK PROXY RESPONSE 72
TABLE 37: PARAMETERS FOR SSH REQUEST 73
TABLE 38: SSH AUTHORIZATION KEY 73
TABLE 39: PARAMETERS FOR STATUS RESPONSE 74
TABLE 40: WIFI STATE ENUMERATION 74
TABLE 41: PARAMETERS FOR WIFI ACCESS POINT REQUEST 75
TABLE 42: PARAMETERS FOR WIFI ACCESS POINT RESPONSE 75
TABLE 43: PARAMETERS FOR WIFI CONNECT REQUEST 76
TABLE 44: WIFI AUTHENTICATION TYPES ENUMERATION 76
TABLE 45: PARAMETERS FOR WIFI CONNECT COMMAND 76
TABLE 46: PARAMETERS FOR WIFI FORGET REQUEST 77
TABLE 47: PARAMETERS FOR WIFI FORGET RESPONSE 77

TABLE 48: PARAMETERS FOR WIFI IP ADDRESS RESPONSE 78
TABLE 49: PARAMETERS FOR WIFI SCAN RESPONSE 79
TABLE 50: PARAMETERS ACCESS POINT STRUCTURE 79
TABLE 51: THE CLOUD SERVICES CONFIGURATION FILE 80
TABLE 52: THE OWNERS ACCOUNT INFORMATION 86
TABLE 53: THE ROBOT SETTINGS 87
TABLE 54: THE ROBOT LIFETIME STATS SCHEMA 87
TABLE 55: MANIFEST.INI META SECTION 91
TABLE 56: MANIFEST.INI IMAGE STREAM SECTIONS 92
TABLE 57: UPDATE-ENGINE STATUS FILE 92
TABLE 58: COMMON ACRONYMS AND ABBREVIATIONS 99
TABLE 59: GLOSSARY OF COMMON TERMS AND PHRASES 100
TABLE 60: TOOLS USED BY ANKI 102
TABLE 61: TOOLS THAT CAN BE USED TO ANALYZE AND PATCH VECTOR 104
TABLE 62: THE SYSTEM FAULT CODES 105
TABLE 63: OTA UPDATE-ENGINE STATUS CODES 105
TABLE 64: THE FILE SYSTEM MOUNT TABLE 107
TABLE 65: THE PARTITION TABLE 107
TABLE 66: FILES 109
TABLE 67: NAMED /DEV DEVICE FILES 109
TABLE 68: THE BLUETOOTH LE SERVICES 110
TABLE 69: THE CUBE'S DEVICE INFO SETTINGS 110
TABLE 70: CUBE'S ACCELEROMETER SERVICE CHARACTERISTICS 111
TABLE 71: VECTOR'S BLUETOOTH LE SERVICES 111
TABLE 72: THE VECTOR'S DEVICE INFO SETTINGS 111
TABLE 73: VECTOR'S SERIAL SERVICE CHARACTERISTICS 112
TABLE 74: THE SERVERS THAT VECTOR CONTACTS. 113
TABLE 75: THE "HEY VECTOR" PHRASES 114
TABLE 76: THE VECTOR QUESTIONS PHRASES 116

EXAMPLE 1: BLUETOOTH LE ENCRYPTION STRUCTURES 55
EXAMPLE 2: BLUETOOTH LE KEY PAIR 55
EXAMPLE 3: BLUETOOTH LE ENCRYPTION & DECRYPTION KEYS 55
EXAMPLE 4: DECRYPTING A BLUETOOTH LE MESSAGE 56
EXAMPLE 5: ENCRYPTING A BLUETOOTH LE MESSAGE 56
EXAMPLE 6: CHECKING THE MANIFEST.INI SIGNATURE 91
EXAMPLE 7: DECRYPTING THE OTA UPDATE ARCHIVES 92
EXAMPLE 8: DECRYPTING THE OTA UPDATE ARCHIVES WITH OPEN SSL 1.1.0 AND LATER 92

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 many areas of 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 Vector's accessories.

PART II: BASIC OPERATION.

- CHAPTER 5: ARCHITECTURE. A detailed look at Vector's overall software architecture.
- CHAPTER 6: POWER MANAGEMENT. A detailed look at Vector's startup, shutdown and other power management architecture.
- CHAPTER 7: AUDIO INPUT AND OUTPUT
- CHAPTER 8: LCD DISPLAY

PART III: COMMUNICATION.

- CHAPTER 9: COMMUNICATION. A look at the communication stack in Vector.
- CHAPTER 10: BLUETOOTH LE. The Bluetooth LE protocol that Vector responds to.
- CHAPTER 12: CLOUD. A look at how Vector syncs with remote services.

PART IV: MAINTENANCE.

- CHAPTER 15: SETTINGS, PREFERENCES AND STATISTICS.
- CHAPTER 16: SOFTWARE UPDATES. How Vector's software updates are applied.

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: FAULT AND STATUS CODES. This appendix provides describes the system fault codes, and update status codes.
- APPENDIX D: FILE SYSTEM. This appendix lists the key files that are baked into the system.
- APPENDIX E: BLUETOOTH LE PROTOCOLS. This appendix provides information on the Bluetooth LE interfaces to the companion Cube, and to Anki Vector
- APPENDIX F: SERVERS. This appendix provides the servers that the Anki Vector and App contacts
- APPENDIX H: PHRASES. This appendix reproduces the phrases that the Vector keys off of.
- APPENDIX I: FEATURES. This appendix enumerates the Vector OS "features" that can be enabled and disabled.
- APPENDIX J: PLEO. This appendix gives a brief overview of the Pleo animatronic dinosaur, an antecedent with many similarities.

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.

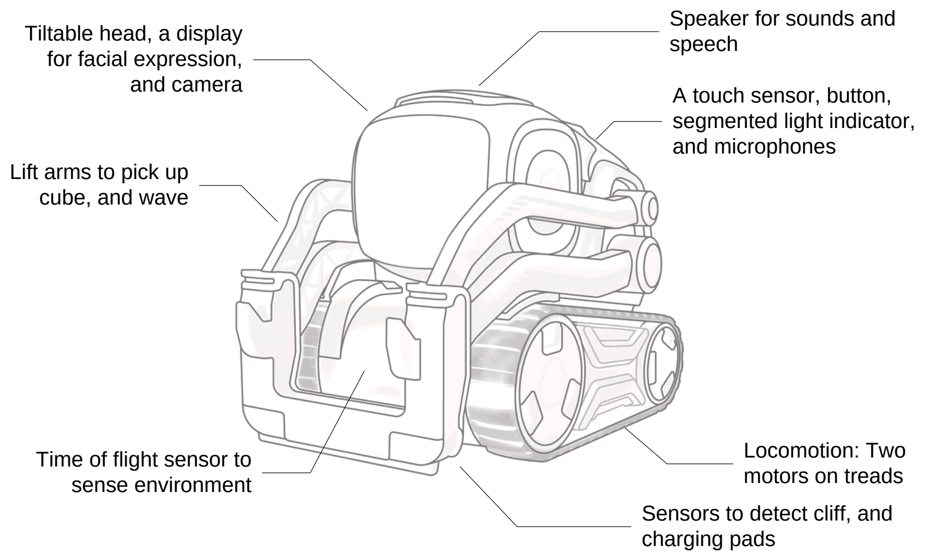

Tiltable head, a display for facial expression, and camera
Lift arms to pick up cube, and wave
Time of flight sensor to sense environment
Speaker for sounds and speech
A touch sensor, button, segmented light indicator, and microphones
Locomotion: Two motors on treads
Sensors to detect cliff, and charging pads

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.

He has a set of operator inputs:

- A touch sensor is used detect petting

[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.

- 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 Vector's 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 *inertial measurement unit* (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?

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

- Vector plans a route to his goals – if he knows where his goal is – along a path free of obstacles; he adapts, moving around in changing conditions.
- A*, Rapidly-Expanding Random Tree (RRT), D*-lite
- Paths are represented as arcs, line segments, and turn points

Mapping and Navigation:

- Maps are built using *simultaneous location and mapping* (SLAM) algorithms, using the camera and IMU gyroscope movement tracking, time of flight sensor to measure distances, and particle system algorithms to fill in the gaps.
- The maps are represented uses quad-tree (position, pose)

Behaviour system:

- Variety of behaviors animations
- Goals, linking up to the guidance system to accomplish them
- A simple emotion model to drive selection of behaviours

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 contact

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 Cozmo's hardware. Vector's software architecture also borrows from Cozmo, and extends it greatly. Andrew Stein was Cozmo's original (but not only) software developer. Brian Chapados's team developed the Android and iOS applications.

Many of Vector's behaviours, senses, and functions were first implemented in Cozmo (and/or in the smartphone application). One notable difference is that Cozmo did not include a microphone.

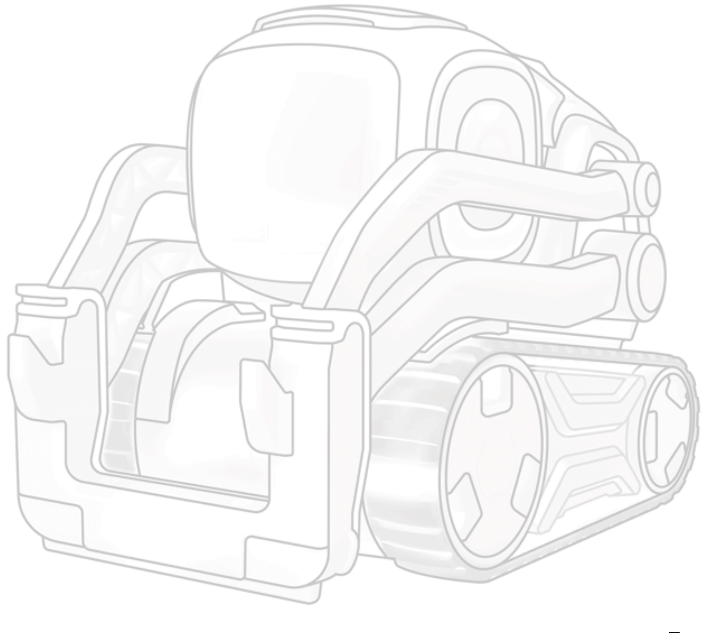
[This page is intentionally left blank for purposes of double-sided printing]

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 Vector's 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.

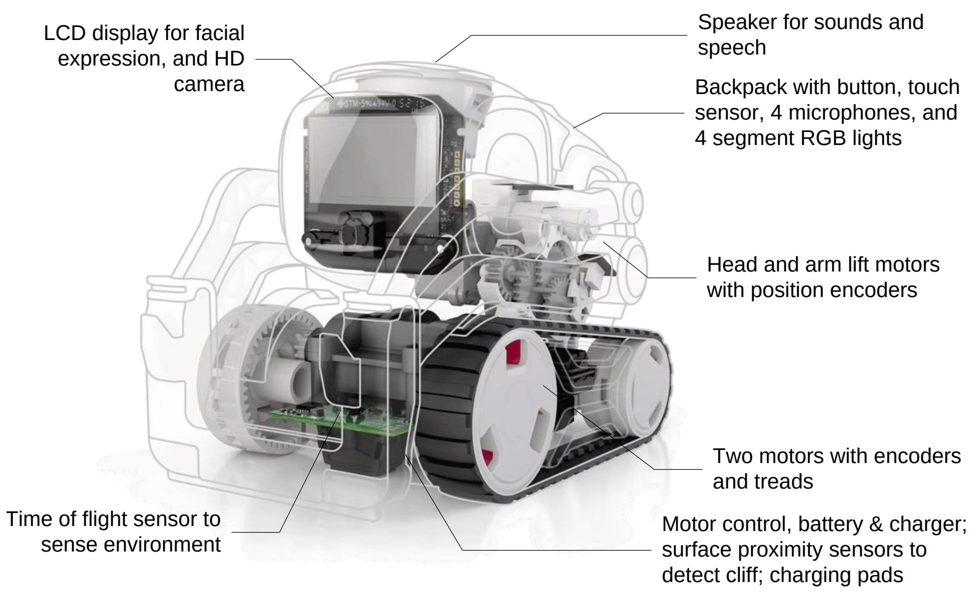

LCD display for facial expression, and HD camera
Speaker for sounds and speech
Backpack with button, touch sensor, 4 microphones, and 4 segment RGB lights
Head and arm lift motors with position encoders
Two motors with encoders and treads
Time of flight sensor to sense environment
Motor control, battery & charger; surface proximity sensors to detect cliff; charging pads

***Figure 2:** Vector’s main elements*

Vector's functional elements are:

***Table 1:** Vector's main elements*

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.

Vector has 6 circuit boards

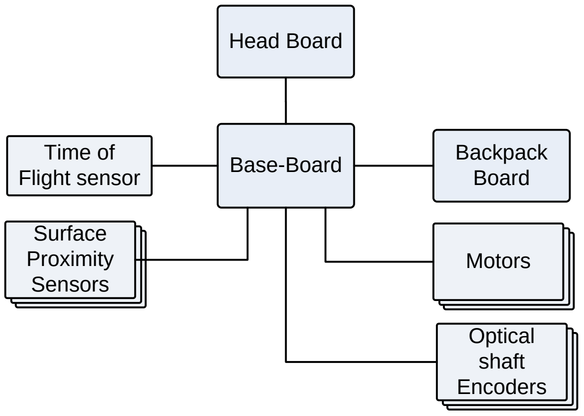


Figure 3: Circuit board topology

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

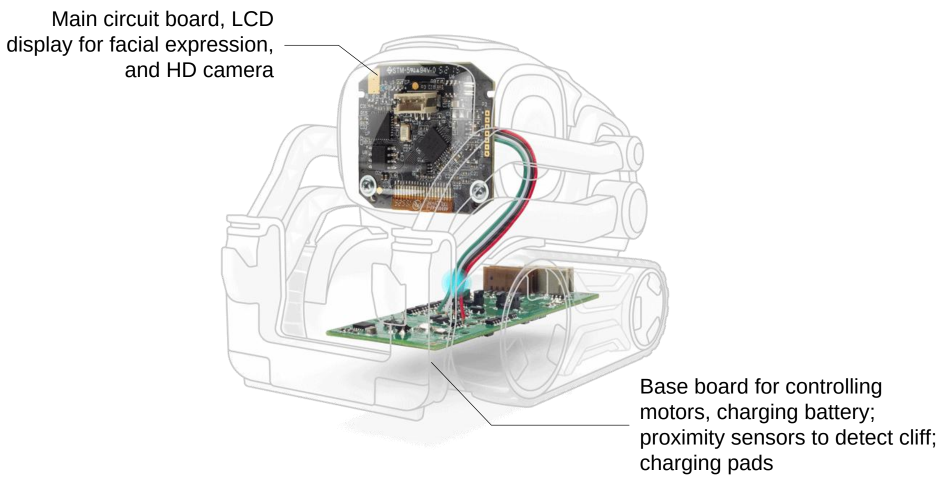

Main circuit board, LCD display for facial expression, and HD camera
Base board for controlling motors, charging battery; proximity sensors to detect cliff; charging pads

Figure 4: Vector's main microcontroller circuit boards

The table below summarizes the boards:

Table 2: Vector's circuit 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 Vector's front.

5.1. POWER SOURCE AND DISTRIBUTION TREE

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

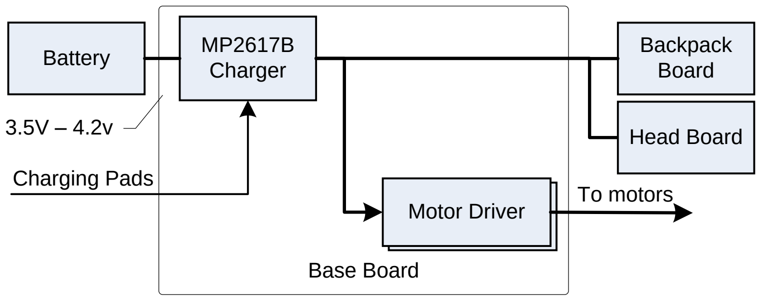


Figure 5: *Power distribution*

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:

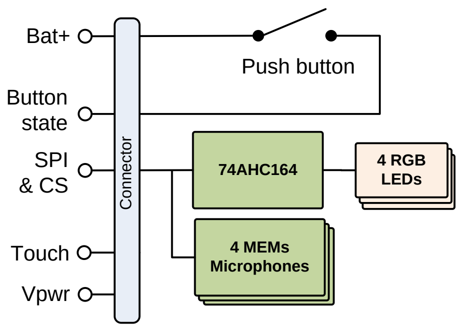


Figure 6: *Backpack board block diagram*

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.

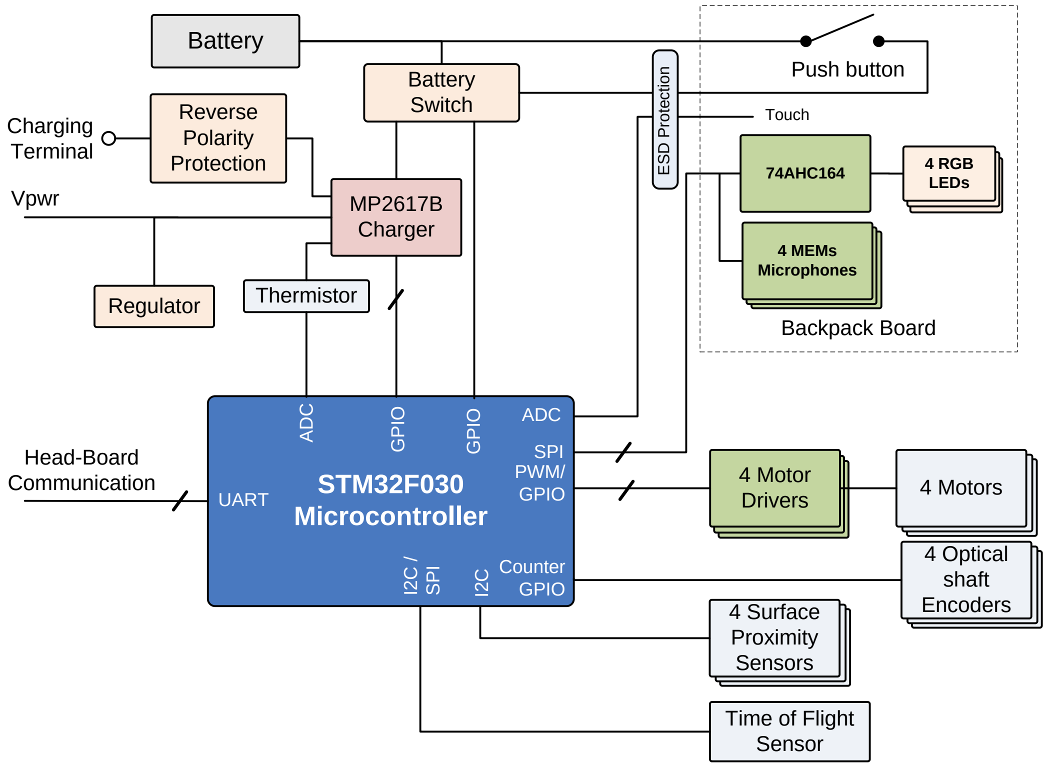


Figure 7: *Base-board block diagram*

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:

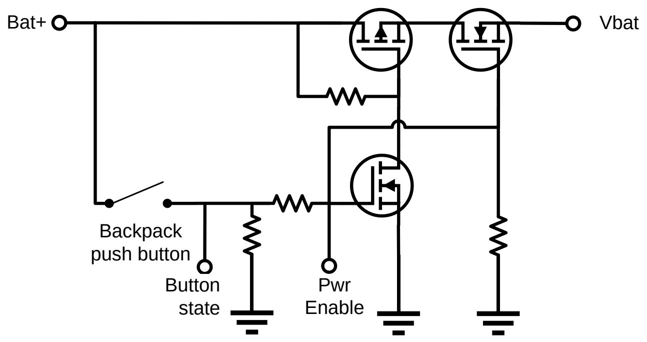


***Figure 8:** A representative battery connect switch*

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
[8] Q14

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

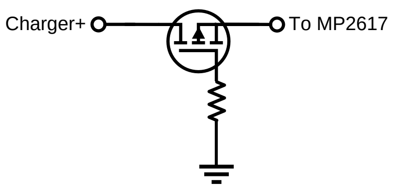


***Figure 9:** A representative PFET based reversed polarity protection*

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.

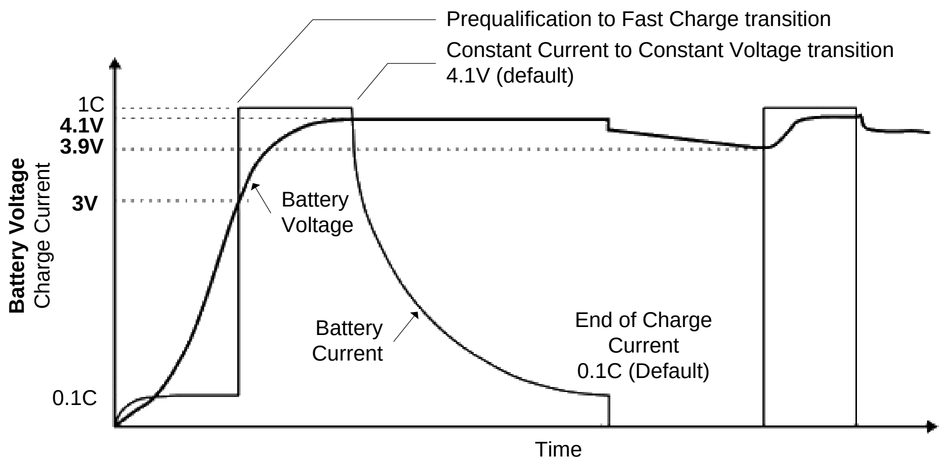


***Figure 10:** Charging profile* (adapted from Texas Instruments)

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 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] 1Ω on the USB power. I tried 1Ω -14Ω; these should have limited the current to 1A and 500mA respectively. Instead, Vector would only pull 40mA - 370mA; in many cases, not enough to 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.

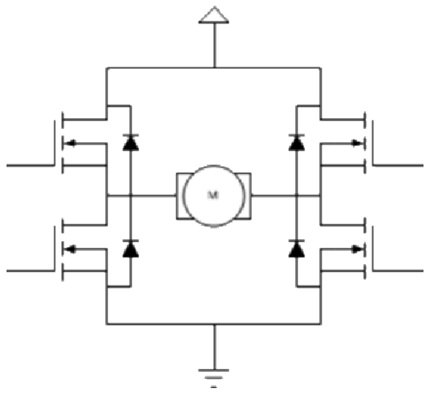


Figure 11: *Motor driver H-bridge*

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 (460800 bits/sec[14]). 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.

[14] Value from the startup logs. Melanie T measured it on an oscilloscope and estimated it to be 2Mbps.

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.

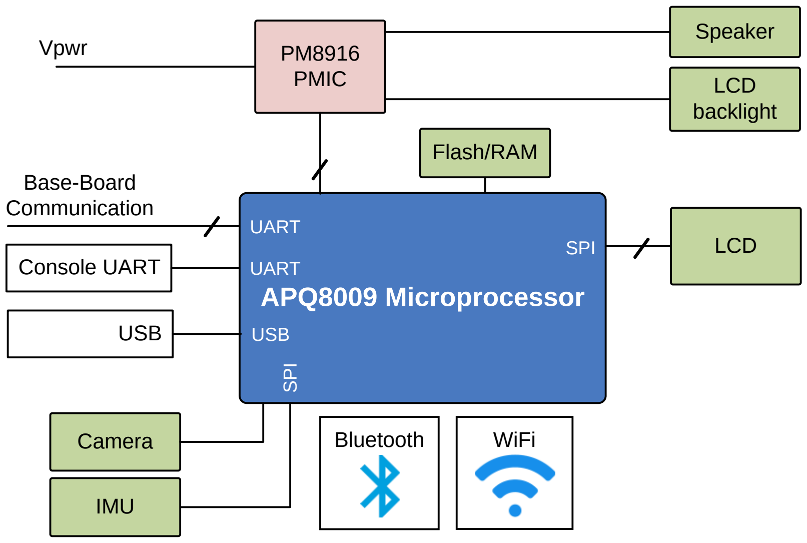


Figure 12: *Head-board block diagram*

The head-board's functional elements are:

Table 5: *The head-boards functional elements*

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

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 camera 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.

US Patent 20372659

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 calibrated
- The IMU is calibrated
- The motor power is calibrated[15]

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 at least one 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.

The USB interface is used to load firmware. The microprocessors include a built in bootloader (ABOOT), which includes support for loading firmware into the devices flash.

For the other functions, 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 have been loaded into the unit

[15] 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?

Bluetooth LE is very unlikely due to the mismatch between its capabilities and the data rates necessary for video used to calibrate the camera.

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

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

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.

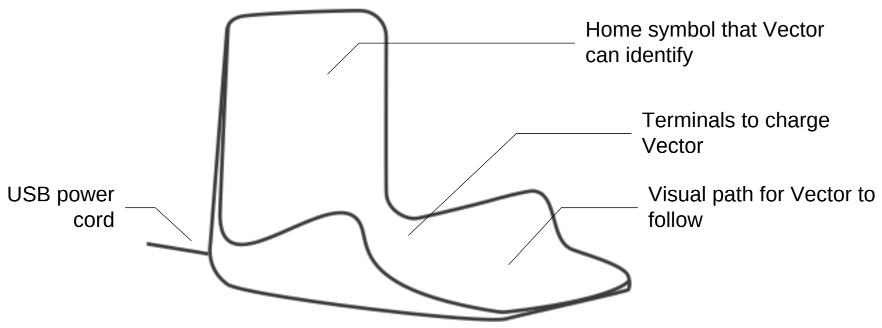

Home symbol that Vector can identify
Terminals to charge Vector
USB power cord
Visual path for Vector to follow

***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).

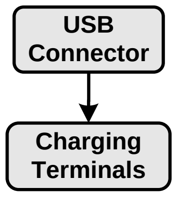


***Figure 14:** Charging station block diagram*

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).

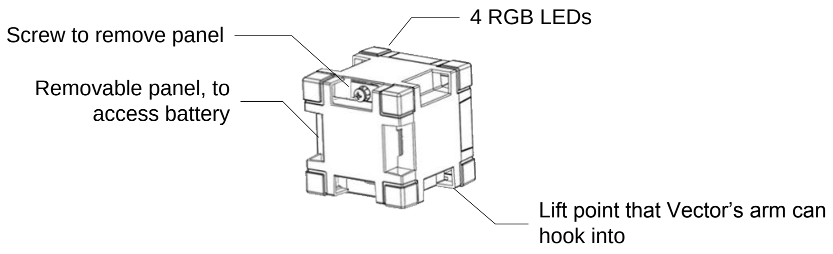

Screw to remove panel
4 RGB LEDs
Removable panel, to access battery
Lift point that Vector's arm can hook into

***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:

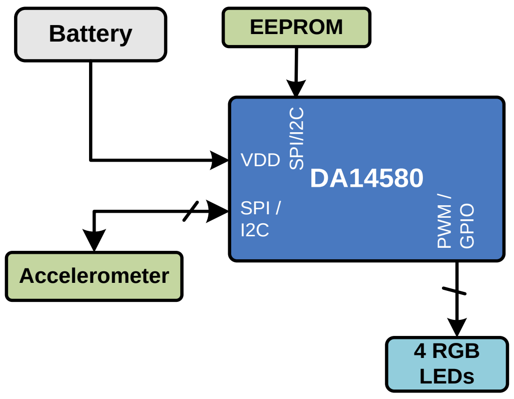


***Figure 16:** Block diagram of the Cube's electronics*

The Cube's electronic design includes the following elements:

***Table 6:** The Cube's electronic design 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.[16]
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

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

	have independent colors.

The communication protocol is given in Appendix E.

11.1. OVER THE AIR FIELD UPDATES

The DA14580 has a minimal ROM boot loader that initializes hardware, moves a secondary boot loader 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 boot loader 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 boot loader – or there isn't a valid application in EEPROM –a new application can be downloaded. The boot loader uses a different set of services and characteristics to support the boot loading 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

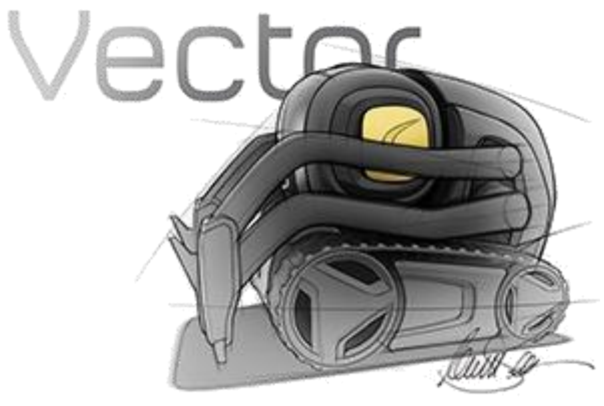
[This page is intentionally left blank for purposes of double-sided printing]

PART II

Basic Operation

This part provides an overview of Vector's software design.

- THE SOFTWARE ARCHITECTURE. A detailed look at Vector's overall software architecture and main modules.
- POWER MANAGEMENT. A detailed look at Vector's startup, shutdown and other power management architecture.
- AUDIO INPUT AND OUTPUT.



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

CHAPTER 5

Architecture

This chapter examines Vector's software architecture:

- The architecture
- The emotion-behaviour system
- The communication infrastructure
- Internal support

12. OVERVIEW OF VECTOR'S COMMUNICATION INFRASTRUCTURE

Vector's architecture has a structure something like:

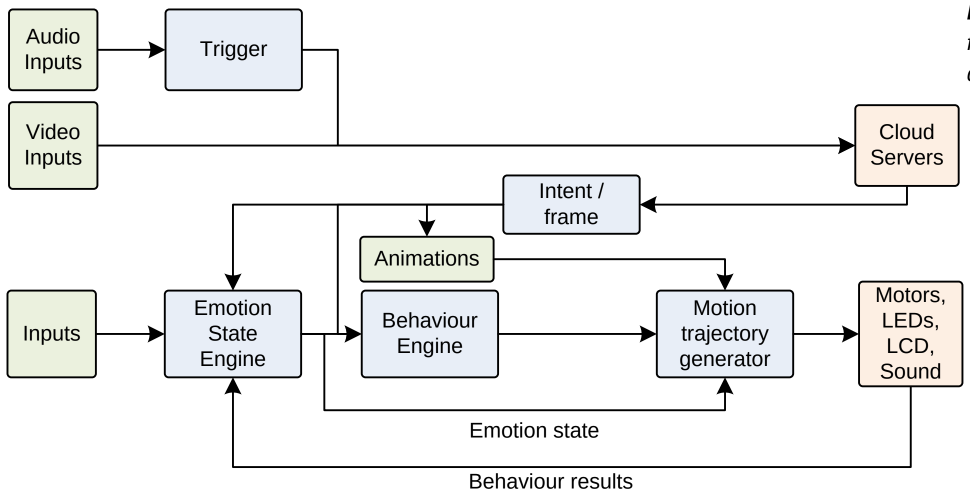


Figure 17: *The overall functional block diagram*

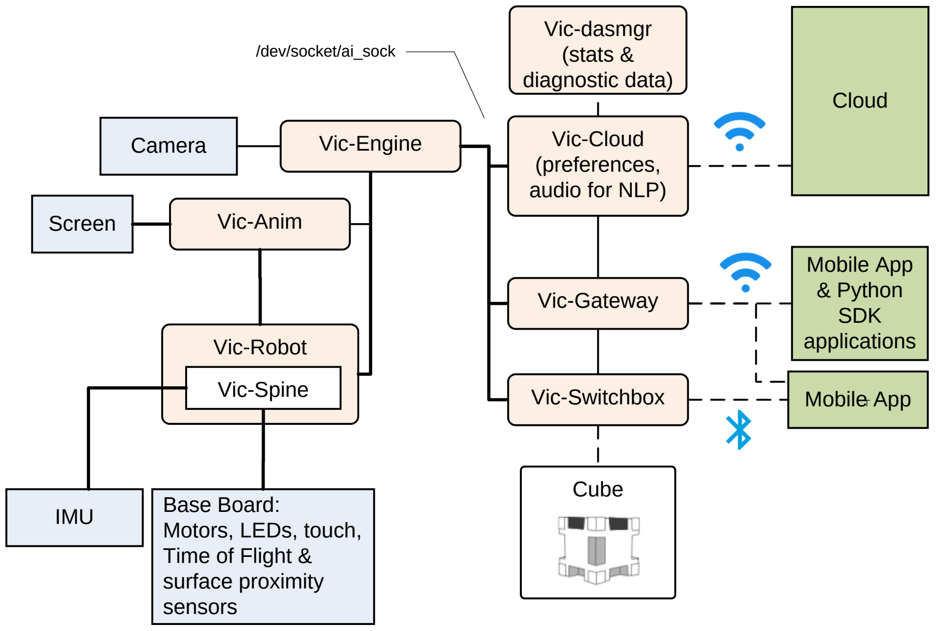
Fast control loops — to respond quickly — are done on the Vector's hardware. Other items, processing heavy including (but not limited to) speech recognition, natural language processing, and training for faces are sent to the cloud.

Vector is built on a version of Yocto Linux. Anki selected this for a balance of reasons: some form of Linux is required to use the Qualcomm processor, the low up front (and royalty) costs, the availability of tools and software modules. The Qualcomm is a multi-processor, with four main processing cores and a GPU. Vector runs a handful of different application programs, in addition to the OS's foundational service tasks and processes.

explored in Casner, *and* Wiltz

12.1. APPLICATION SERVICES ARCHITECTURE

The application is divided into the following services



/dev/socket/ai_sock
Vic-dasmgr (stats & diagnostic data)
Cloud
Camera
Vic-Engine
Vic-Cloud (preferences, audio for NLP)
Screen
Vic-Anim
Vic-Gateway
Mobile App & Python SDK applications
Vic-Robot
Vic-Spine
Vic-Switchbox
Mobile App
IMU
Base Board: Motors, LEDs, touch, Time of Flight & surface proximity sensors
Cube

Figure 18: *The overall communication infrastructure*

There are multiple applications that run:

Table 7: *Vector processes*

Services	Speculated purpose
vic-anim	Probably plays multi-track animations (which include motions as well as LCD display and sound) config file: /anki/etc/config/platform_config.json /anki/data/assets/cozmo_resources/webserver/webServerConfig_anim.json
vic-bootAnim	LCD and sound animations during boot?
vic-cloud	Probably connects to the cloud services for natural language
vic-dasmgr	
vic-engine	The behaviour / emotion engine. Hooks into the camera face recognizer.
vic-gateway	Responsible for the local API/SDK services available as gRPC services on https.
vic-robot	Basic power management. Resets watchdog timer. Internally has “vic-spine” that manages the sensors.
vic-switchboard	Supports the Bluetooth LE communication interface, including the mobile application protocol (see chapter 9). Routes messages between the other services? Manages the access keys

13. ROBOT DATA

13.1. EMR

The EMR partition holds the following information.

- Electronic serial number (ESN). This the same serial number as printed on the bottom of Vector
- Hardware FER (?)
- Model number
- Lot code
- Playpen ready flag
- Playpen passed flag
- Packed out flag
- Packed out date

This information is not modified after manufacture; it persists after a device reset or wipe.

14. REFERENCES & RESOURCES

Casner, Daniel, *Consumer Robots from Smartphone SoCs*, Embedded Systems Conference Boston, 2019 May 15
https://schedule.esc-boston.com//session/consumer-robots-from-smartphone-socs/865645

Stein, Andrew; Kevin Yoon, Richard Alison Chaussee, Bradford Neuman, Kevin M.Karol, US Patent US2019/01563A1, *Custom Motion Trajectories for Robot Animation*, Anki, filed 2018 Jul 13, published 2019 Apr 18,

Qualcomm, *Snapdragon™ 410E (APQ8016E) r1034.2.1 Linux Embedded Software Release Notes*, LM80-P0337-5, Rev. C, 2018 Apr 10
lm80-p0337-5_c_snapdragon_410e_apq8016e_r1034.2.1_linux_embedded_software_revc.pdf

Wiltz, Chris, *Lessons After the Failure of Anki Robotics,* Design News, 2019 May 21
https://www.designnews.com/electronics-test/lessons-after-failure-anki-robotics/140103493460822

CHAPTER 6

Power management

This chapter is about Vector's power management:

- The startup process
- The shutdown steps
- The battery management
- Charger info?

Vector, in the advertising material, aspires to be on all the time. In practice, he will be turned on to play with, turned off when done playing – or when his battery runs low, or before being shipped.

15. STARTUP

His startup is based on the Android boot loader and linux startup.[17] These are otherwise not relevant to Vector, and their documentation is referred to. The boot process gets quite far before knowing why it booted up, or being able to response in complex fashion.

1. The backpack button is pressed, or Vector is placed into the charger. This powers the base board, and the head-boards.
2. The base-board displays an animation of the backpack LEDs while turning on. If turned on from a button press, may not fully commit. If the button is released before the LED segments are fully lit, the power will go off. If the held that long, the base-board keeps closes the battery switch closed.

Meanwhile, on the head-board:

1. "Qualcomm's Primary Boot Loader is verified and loaded into [RAM] memory[18] from BootROM, a non-writable storage on the SoC. [The primary boot loader] is then executed and brings up a nominal amount of hardware, then verifies the signature of the next boot loader in the chain, loads it, [and] then executes it." Nolen Johnson
2. "The next boot loader(s) in the chain are SBL*/XBL (Qualcomm's Secondary/eXtensible Boot Loader). These early boot loaders bring up core hardware like CPU cores, the MMU, etc. They are also responsible for bringing up core processes .. [for] TrustZone. The last purpose of SBL*/XBL is to verify the signature of, load, and execute aboot/ABL [Android boot loader]."
   The Android boot loader (ABoot) is stored on the "`ABOOT`" partition.
3. ABoot can either program the flash with firmware via boot loader mode, or load a kernel. The kernel can be flagged to use a recovery RAM disk, or mount a regular system.

[17] An ideal embedded system has a fast (seemingly instant) turn on. Vector's startup *isn't* fast. The steps to check the data integrity of the large flash storage – including checking the security signatures – and the complex processes that linux provides each contribute to the noticeable slow turn on time. Checking the signatures is inherently slow, by design.
[18] The boot loader is placed into RAM for execution to defeat emulators.

4. If recovery mode, it will load the kernel and file systems from the RECOVERY partitions.
5. ABoot loads the kernel and RAM file system from the "BOOT_A" partition and passes it command line to perform a check of the boot and RAM file system the signatures.[19]
6. The "RAM file system [is] mounted at the root path during the Linux initialization. It contains init, the first user space process. Among its duties, init triggers the partition mounts. dm-verity then verifies the integrity of specified partitions under fstab (e.g. the system partition)." Roee Hay
    a. Vector's kernel does not have the dm-verity (device mapper verity) feature was enabled, nor are any of the files systems in fstab marked for verity checking.

Note: The keys for the boot loaders and TrustZone are generated by Qualcomm, with the public keys programmed into the hardware fuses before delivery to Anki or other customers. The signed key pair for the secondary boot loader is not necessarily the same signed key pair for the ABOOT. They are unique for each of Qualcomm's customer. Being fuses they cannot be modified, even with physical access.

15.1. REGULAR SYSTEM BOOT

The regular boot uses systemd to allow of the startup steps to be performed in parallel. The rough start up sequence is:

1. Starts the Qualcomm Secure Execution Environment Communicator (dev-qseecom.device) and ION memory allocator (dev-ion.device)
2. The encrypted user file system is checked, and mounted (via the mount-data service). This filesystem is where the all of the logs, peoples faces, and other information specific to the individual Vector are stored. The keys to this file system are stored in the TrustZone in the MPU's SOC fuse area. This file system can only be read by the MPU that created it.
3. After that several mid-layer communication stacks are started:
    a. usb-service any time after that
    b. the WiFi connection manager (connman)
    c. The time client (chronyd), to retrieve network time. (Vector does not have a clock that keeps time when turned off)
    d. init-debuggerd
4. multi-user, sound, init_post_boot
5. The "Victor Boot Animator" is start (~8 seconds after power on). This is probably the sparks.
6. Victor Boot completes ~20.5 after power on, and the post boot services launches
7. vic-crashloader
8. vic-robot
9. Once the startup has sufficiently brought up enough the next set of animations the sound of boot

[19] The check specifies the blocks on the storage to perform a SHA256 check over, and provides expected signature result.

10. VicOS is running ~32 seconds after power on. The boot is complete and Vector is ready to play

15.2. ABNORMAL SYSTEM BOOT

If there is a problem during startup – such as one of the services is unable to successfully start, the a fault code is stored in TBD and the faultCode displayed. See Appendix C for fault codes.

16. SHUTDOWN

- Turning Vector off manually
- Vector turning off spontaneously due to brown-out or significant loss of power
- Vector turning off (under low power) by direction of the head-board

Vector cannot be turned off via Bluetooth LE, or the local gRPC SDK access. There are no exposed commands that do this. Using a verbal command, like "turn off" does not direct Vector to shut off (disconnect the battery). Instead it goes into a quiet mode. Although it may be possible for a Cloud command to turn Vector off, this seems unlikely.

However, there is likely a command to automate the manufacture and preparation for ship process.

16.1. TURNING VECTOR OFF (INTENTIONALLY)

Tracks the reason for shutdown:

***Table 8:** Vector shutdown codes*

Element	Description & Notes
SHUTDOWN_BATTERY_CRITICAL_TEMP	Vector shut down automatically because the battery temperature was too high.
SHUTDOWN_BATTERY_CRITICAL_VOLT	Vector shut down automatically because the battery voltage was too low.
SHUTDOWN_BUTTON	Vector was shut down by a long button press.
SHUTDOWN_GYRO_NOT_CALIBRATING	Vector shut down automatically because of an IMU problem(?)
SHUTDOWN_UNKNOWN	Vector shut down unexpectedly; the reason is not known. Likely a brown-out or battery voltage dipped low faster than Vector could respond to.

It is not clear where the shutdown code is stored

16.2. UNINTENTIONALLY

The base-board is responsible for keeping the battery connected. However brownouts, self-protects when the voltage get to too low, and bugs can cause the battery to be disconnected.

16.3. GOING INTO AN OFF STATE

When Vector wants to intentionally turn off, it cleans up its state, to gracefully shutdown the linux system and tells the base-board to disconnect the battery.

17. POWER MANAGEMENT

17.1. BATTERY MANAGEMENT

There isn't a coulomb counter to track the remaining energy in the battery. At the broadest strokes, the battery voltage is used to predict the battery state of charge.

17.1.1 Battery levels

Vector maps the battery voltage into a battery level, taking into account whether or not the charger is active:

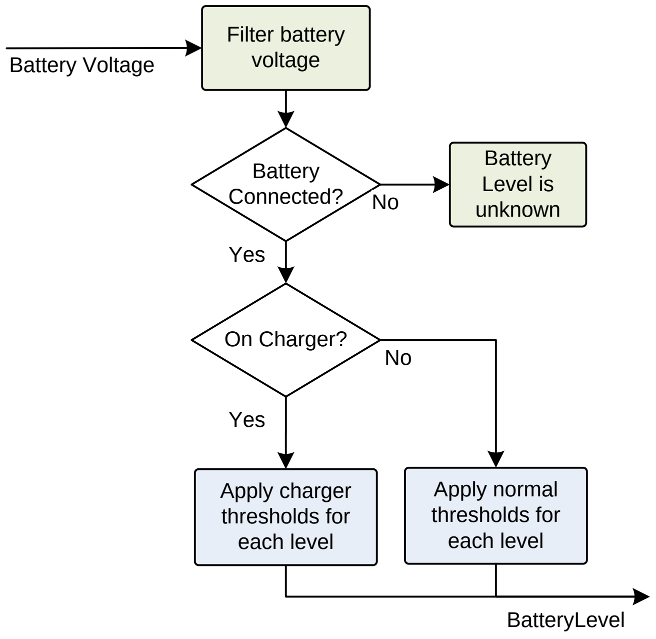


***Figure 19:** The battery level classification tree*

Note: The battery voltage is filtered because the voltage will bounce around with activity by the motors, driving the speaker and processors.

The BatteryLevel enumeration is used to categorize the condition of the Cube and Vector batteries:

***Table 9:** BatteryLevel codes[20] as they apply to Vector*

Element	Description & Notes
BATTERY_LEVEL_FULL	Vector's battery is at least 4.1V
BATTERY_LEVEL_LOW	Vector's battery is 3.6V or less; or if Vector is on the charger, the battery voltage is 4V or less.
BATTERY_LEVEL_NOMINAL	Vector's battery level is between low and full.
BATTERY_LEVEL_UNKNOWN	If the battery is not connected, Vector can't measure its battery. This enumeration is (probably) reserved for the case where the Cube is not in contact, thus its battery level is not known

[20] The levels are from robot.py

The battery levels are organized conventionally:

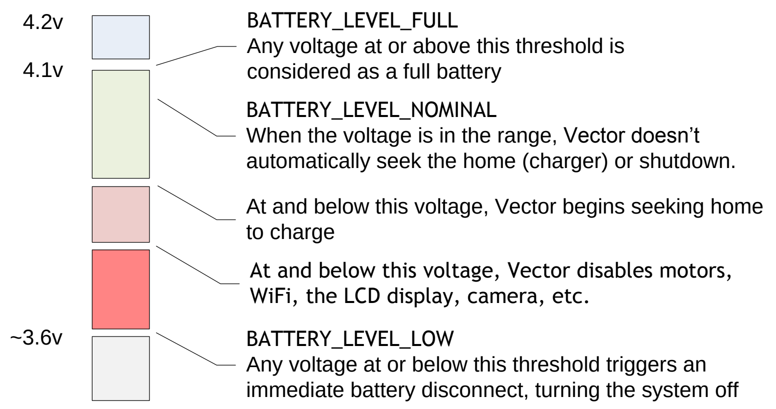

4.2v
4.1v
~3.6v
BATTERY_LEVEL_FULL
Any voltage at or above this threshold is considered as a full battery
BATTERY_LEVEL_NOMINAL
When the voltage is in the range, Vector doesn't automatically seek the home (charger) or shutdown.
At and below this voltage, Vector begins seeking home to charge
At and below this voltage, Vector disables motors, WiFi, the LCD display, camera, etc.
BATTERY_LEVEL_LOW
Any voltage at or below this threshold triggers an immediate battery disconnect, turning the system off

Figure 20: *The battery thresholds*

17.1.2 Battery Statistics

The battery management collects a set of battery related statistics and state information, with named key-value pairs.

Table 10: *Battery statistics, possibly posted to DAS*

Element	Description & Notes
battery.battery_level_changed	This is set when the battery level has changed from event posting.
battery.cooldown	Indicates that Vector is or needed to pause charging and activity to let the battery cool down.
battery.encoder_power_stats	?? Strange name. Is this the voltage seen on the charger input and charging duration stats?
battery.fully_charged_voltage	The battery voltage seen when the charger reported the battery to be fully charged.
battery.is_charging_changed	This is set when the state of charging has changed from event posting.
battery.on_charger_changed	
battery.saturation_charging	
battery.temp_crossed_threshold	
battery.temperature_stats	Information about the range of battery temperatures that have been observed; e.g. min/max, average, etc.
battery.voltage_reset	
battery.voltage_stats	Information about the range of battery voltages that have been observed; e.g. min/max, average, etc.

{commentary: Are these posted to DAS?}

17.2. RESPONSES, SHEDDING LOAD / POWER SAVING EFFORTS

Vector's main (power-related) activity modes are:

- active, interacting with others

- calm, where primarily sitting still, waiting for assistance or stimulation
- sleeping

Depending on the state of the battery – and charging – Vector may engage in behaviours that override others.

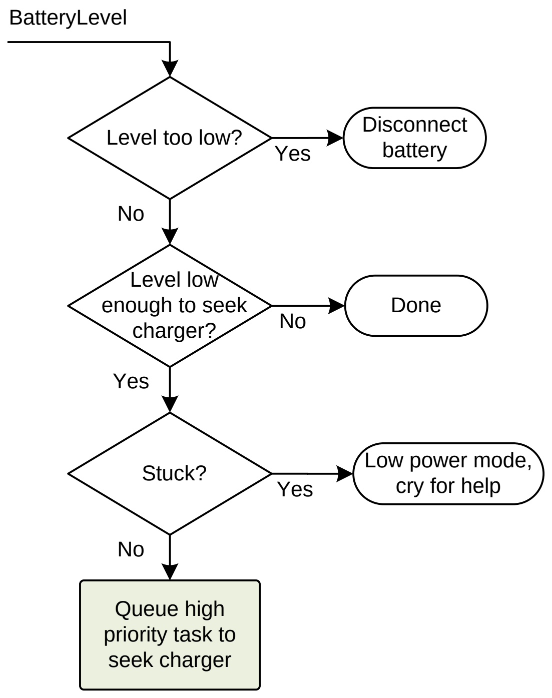

BatteryLevel
Level too low?
Yes
Disconnect battery
No
Level low enough to seek charger?
No
Done
Yes
Stuck?
Yes
Low power mode, cry for help
No
Queue high priority task to seek charger

Figure 21: *The response to battery level*

If Vector is unable to dock (or even locate a dock) he sheds load as he goes into a lower state:

- He no longer responds to his trigger word or communicates with WiFi servers
- He turns off camera and LCD; presumable the time of flight sensor as well.
- He reduces processing on the processor

18. RESOURCES

Android, *Verified Boot*
https://source.android.com/security/verifiedboot/

https://gitlab.com/cryptsetup/cryptsetup/wikis/DMVerity

Johnson, Nolen, *Qualcomm's Chain of Trust,* Lineage OS, 2018 Sept 17
https://lineageos.org/engineering/Qualcomm-Firmware/

A good overview of Qualcomm's boot loader, boot process, and differences between versions of Qualcomm's process. Quotes are slightly edited for grammar.

Qualcomm, *DragonBoard™ 410c based on Qualcomm® Snapdragon™ 410E processor Little Kernel Boot Loader Overview,* LM80-P0436-1, Rev D, 2016 Jul
lm80-p0436-1_little_kernel_boot_loader_overview.pdf

Hay, Roee. *fastboot oem vuln: Android Bootloader Vulnerabilities in Vendor Customizations*, Aleph Research, HCL Technologies
https://www.usenix.org/system/files/conference/woot17/woot17-paper-hay.pdf

CHAPTER 7

Audio

This chapter is about the sound input and output system:

- The audio input
- The audio filtering, and triggering of the speech recognition
- The audio output
- Text to speech

19. AUDIO INPUT

The audio input is used to both give Vector verbal interaction, and to give him environmental stimulation:

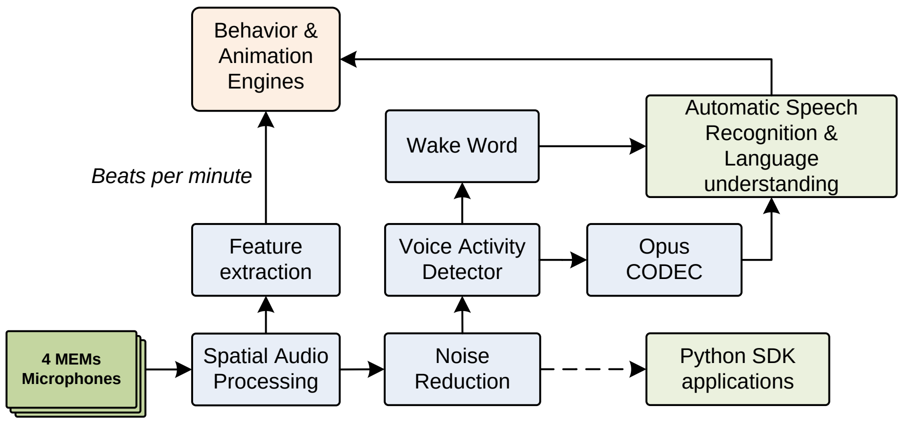


Figure 22: *The audio input functional block diagram*

- Spatial audio processing localizes the sound of someone talking from the background music.
- The feature extraction, detects the tempo of the music. If the tempo is right, Vector will dance to it. It also provides basic stimulation to Vector.
- Noise reduction, makes for the best sound.
- Voice activity detector, usually triggered off of the signal before the beam-forming.
- A wake word is used to engage the automatic speech recognition system.
- The speech recognition system is on a remote server. The audio sent to the automatic speech recognition system is compressed to reduce data usage.

The responsibility for these functions is divided across multiple processes and boards in Vector:

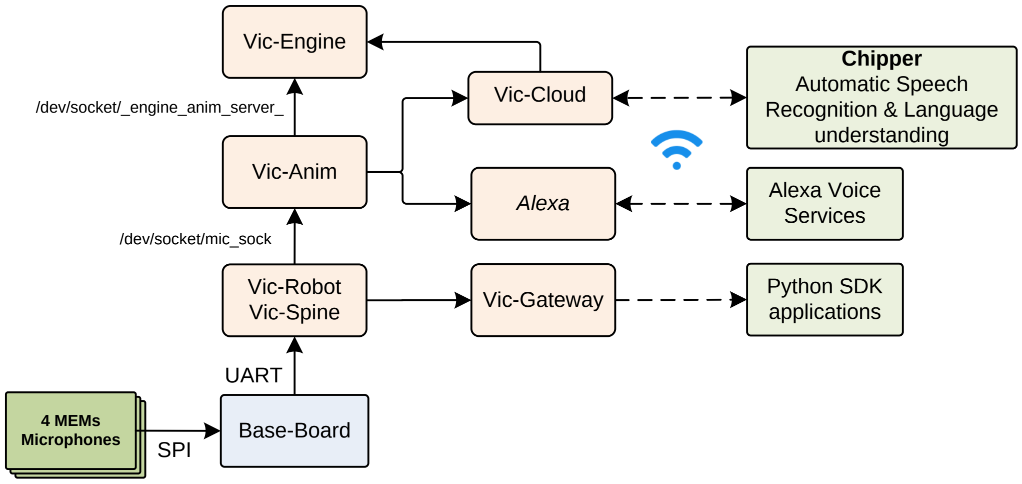


***Figure 23:** The audio input architecture*

Note: providing the audio input to the SDK (via Vic-gateway) was never completed. It will be discussed based on what was laid out in the protobufs specification files.

The audio processing blocks, except where otherwise discussed, are part of Vic-Anim. These blocks were implemented by Signal Essence, LLC. They probably consulted on the MEMs microphones and their configuration. The Qualcomm family includes software support for these tasks, as part of the Hexagon DSP SDK; it is not known how much of this Signal Essence took advantage off.

19.1. THE MICROPHONES

The microphone array is 4 MEMs microphones that sample the incoming sound and transfer the samples to base-board. The audio is sampled by the base-board at 15,625 samples/sec. The audio is transferred to the Vic-spine module (part of Vic-robot) in regular communication with the head-board. The audio samples are extracted and forward to the Vic-Anim process.

The audio samples, once received, are processed at 16,000 samples/sec. ("As a result the pitch is altered by 2.4%") The signal processing is done in chunks of 160 samples.

Note: the Customer Care Information Screen (CCIS) shows the microphones to be about 1024 when quiet. If this is center, the max would be 2048… or 11 bit. Probably is 12 bit.

19.2. SPATIAL AUDIO PROCESSING

The spatial audio processing is uses multiple microphones to pick-out the wanted signal and cancel out the unwanted. Note: The spatial audio processing is bypassed until voice activity has been detected.

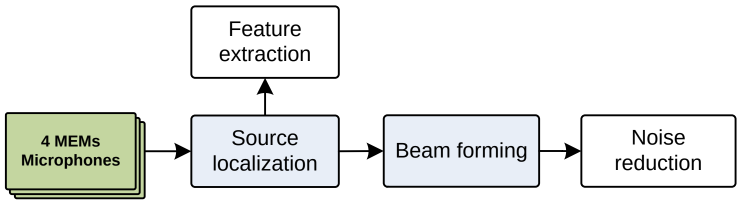


***Figure 24:** The spatial audio processing*

THE SOURCE LOCALIZATION estimates direction of arrival of the person talking.

BEAM-FORMING combines the multiple microphone inputs to cancels audio coming from other directions.

The output of this stage includes:

- A histogram of the directions that the sound(s) in this chunk of audio came from. There are 12 bins, each representing a 30° direction.
- The direction that is picked for the origin of the sound of interest
- A confidence value for that direction
- A measure of the background noise

19.3. NOISE REDUCTION

Noise reduction identifies and eliminates noise and echo in the audion input

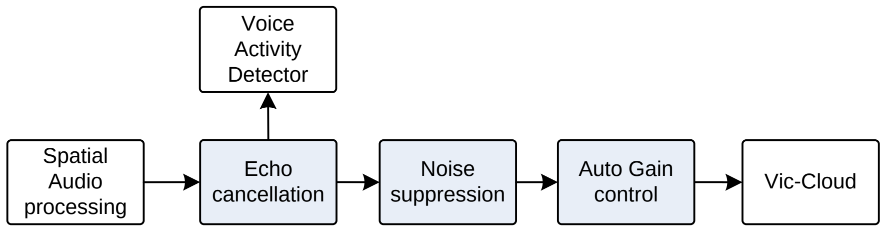


Figure 25: *The audio noise reduction*

ACOUSTIC ECHO CANCELLATION cancels slightly delayed repetitions of a signal.

NOISE SUPPRESSION is used to eliminate noise.

The combination of spatial processing and noise reduction gives the cleanest sound (as compared with no noise reduction and/or no spatial processing).

19.4. VOICE ACTIVITY DETECTOR AND WAKE WORD

The voice activity detector is given cleaned up sound from multiple microphones without beam-forming. When it detects voice activity, then the spatial audio processing is fully enabled.[21]

The voice activity detector and the wake word are used so that downstream processing – the search for the wake word, and the automatic speech recognition system – are not engaged all the time. They are both expensive (in terms of power and data usage), and the speech recognition is prone to misunderstanding.

When the voice activity detector triggers – indicating that a person may be talking – the spatial audio processing is engaged (to improve the audio quality) and the audio signals are passed to the Wake Word Detector.

This detector is provided by Sensory, Inc.[22] Vector's asset files include recognition for "Hey, Vector" and "Alexa." The recognition is locale dependent, detecting different wake words for German, etc. It may be possible create other recognition files for other wake words.

[21] Vector's wake word detection, and speech recognition is pretty hit and miss. Signal Essence's demonstration videos show much better performance.. The differences are they used more microphones and the spatial audio filtering in their demos.
[22] There is also an Alexa wake word detector from Pryon, which appears to be standard for Alexa device SDKs. It is not clear if it is used.

When the wake word is heard,

1. A connection (via Vic-Cloud) is made to the remote speech processing server for automatic speech recognition.
2. A "WakeWordBegin" event message is posted to Vic-Engine and Vic-Gateway. Vic-Gateway may forward the message on to a connected application.

19.5. CLOUD SPEECH RECOGNITION

The audio snippets are sent to a remote server known as "chipper" for processing.

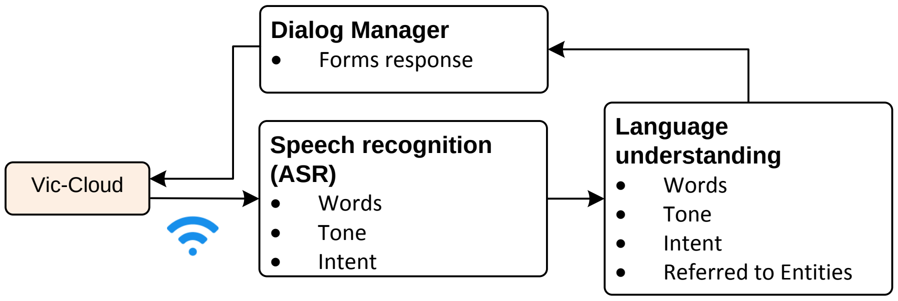


Figure 26: *The speech recognition*

Chipper does speech recognition, and some language understanding. What the user says is mapped to an "user intent" – this may be a command, or a question to be answered. The intent also includes some supporting information – the colour to set the eyes to, for instance. Many of the phrase patterns and the intent they map to can be found in Appendix H. The intent may be further handled by Anki servers; an intent is eventually sent back to Vector.

When the response comes sends a "WakeWordEnd" event message is posted to Vic-Engine and Vic-Gateway. Vic-Gateway may forward the message on to a connected application. This message includes the intent JSON data structure (if any).

19.6. CONNECTIONS WITH VIC-GATEWAY

It is clear that the Anki made provisions to connect the audio stream to Vic-Gateway (and potentially Vic-Cloud) but were unable to complete the features before they ceased operation.

The SDK would have been able to

- Enabled and disabled listening to the microphone(s)
- Select whether the audio would be have the spatial audio filter and noise reduction processing done on it.
- Include the direction of sound information from the spatial audio processing (see section *19.2 Spatial audio processing*)
- 1600 audio samples; Note: this is 10x the chunk size of the internal processing size

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

PART III

Communication

This part provides details of Vector's communication protocol

- 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 interacts with remote services

drawing by Jesse Easley

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

CHAPTER 9

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

20. 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 communication stacks:

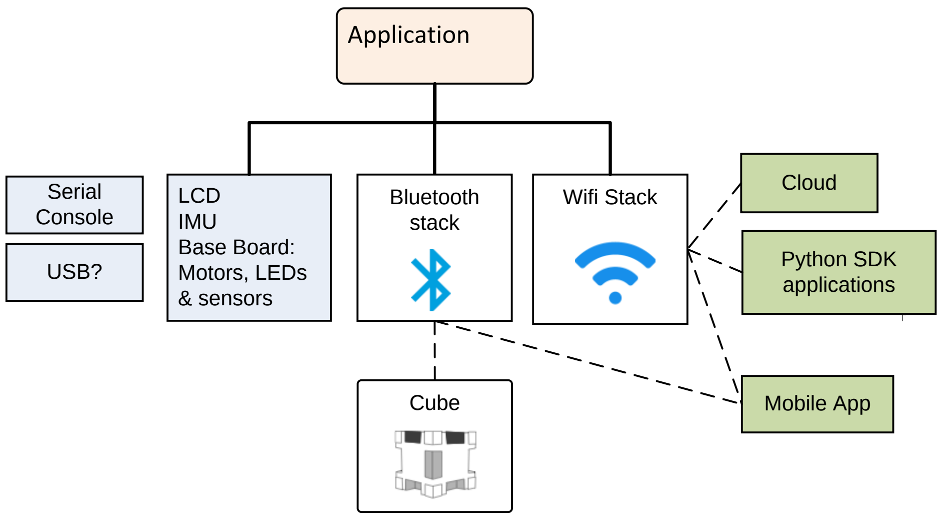


Figure 27: *The overall communication infrastructure*

21. INTERNAL COMMUNICATION WITH PERIPHERALS

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

21.1. COMMUNICATION WITH THE BASE-BOARD

The head board communicates with the base board using a serial interface. The device file is `/dev/ttyHS0`.

Data rate: 460800 bits/sec[23]

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
- A CRC check

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.

The messages are sent fast enough to support microphone sample rate of 15625 samples/second.

21.2. SERIAL BOOT CONSOLE

The head-board includes a 115200, 8 data bits no parity, 1 stop bit; the device file is `/dev/ttyHSL0`. Only prints the boot console. (This is passed in the commanded line by the bootloader)

21.3. 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/initscriptsusb` file enables the USB and the usual functionfs adb. It lives 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.

Vectors log shows the USB being disabled 24 seconds after linux starts.

[23] Value from the startup logs. Melanie T measured it on an oscilloscope and estimated it to be 2Mbps.

22. 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.
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:

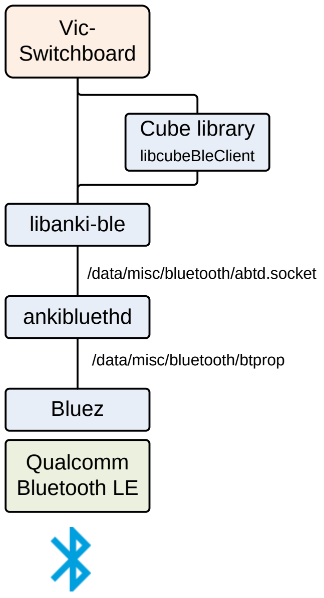


Figure 28: *The Bluetooth LE stack*

The elements of the Bluetooth LE stack include:

Table 11: *Elements of the Bluetooth LE stack*

Element	Description & Notes
ankibluetoothd	A server daemon. The application layer communicates with it over a socket; `/data/misc/bluetooth/abtd.socket`
BlueZ	Linux's official Bluetooth stack, including Bluetooth LE support. The Anki Bluetooth daemon interacts with it over a socket: /data/misc/bluetooth/btprop
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 companion cube.
libcubeBleClient.so[24]	A library to communicate with the companion cube, play animations on its LEDs, detect taps and the orientation of the cube.

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

viccubetool	Probably used to update the firmware in the Cube.

23. COMMUNICATING WITH MOBILE APP AND SDK

Vector's *robot name* is something that looks like "Vector-E5S6". This name is used consistently; it will be Vector's:

- advertised Bluetooth LE peripheral name (although spaces are used instead of dashes)
- mDNS network name (dashes are used instead of spaces),
- the name used to sign certificates, and
- it will be the name of his WiFi Access Point, when placed into Access Point mode

23.1. CERTIFICATE BASED AUTHENTICATION

A *session token* is always provided by Anki servers.[25] It is passed to Vector to authenticate with him and create a client token. The session token is passed to Vector via the Bluetooth LE RTS protocol or the HTTPS-based SDK protocol; Vector will return a client token. The session token is single use only.

A *client token* is passed to Vector in each of the HTTPS-based SDK commands, and in the Bluetooth LE SDK Proxy commands. It is generated in one of two ways. One method is by the Bluetooth LE command (cloud session); the other is by posting "/v1/user_authentication" SDK request. The client token should be saved indefinitely for future use. It is not clear if the client token can be shared between the two transport mechanisms.

A *certificate* is also generated by Vector in the case of the SDK request. The certificate is intended to be added to the trusted SSL certificates before an HTTPS communication session. The certificate issued by Vector is good for 100 years.

The typical information embedded in a Vector certificate:

Table 12: *Elements of a Vector certificate*

Element	**Value**
Common Name	*Vector's robot name*
Subject Alternative Names	*Vector's robot name*
Organization	Anki
Locality	SF
State	California
Country	US
Valid From	*the date the certificate was created*
Valid To	*100 years after the date the certificate was created*
Issuer	*Vector's robot name*, Anki
Serial Number	

[25] https://groups.google.com/forum/#!msg/anki-vector-rooting/YlYQsX08OD4/fvkAOZ91CgAJ
https://groups.google.com/forum/#!msg/anki-vector-rooting/XAaBE6e94ek/OdES50PaBQAJ

CHAPTER 10

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 E.

24. COMMUNICATION PROTOCOL OVERVIEW

Vector advertises services on Bluetooth LE, with the Bluetooth LE peripheral name the same as his robot name (i.e. something that looks like "Vector-E5S6".)

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:

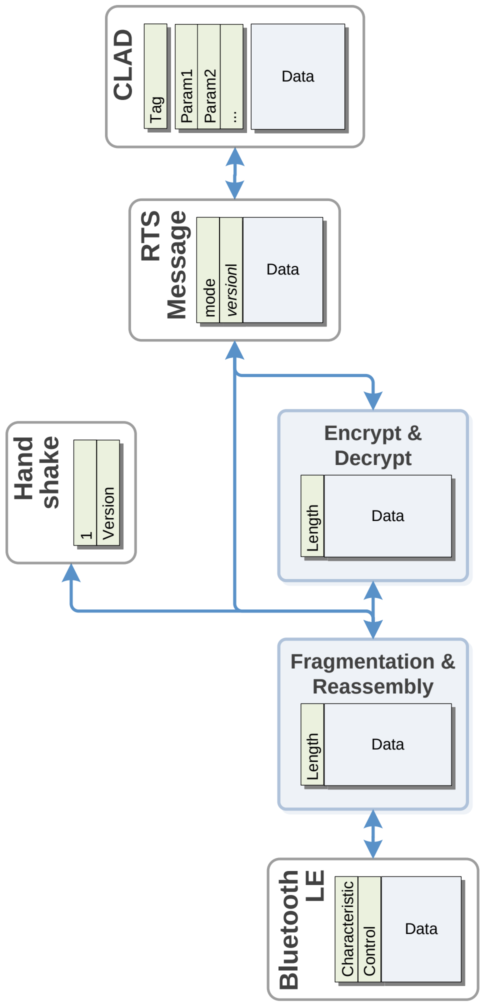


Figure 29: *Overview of encryption and fragmentation stack*

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 E.

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,

24.1. 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.

The process to set up a Bluetooth LE communication with Vector is complex. The sequence has many steps:

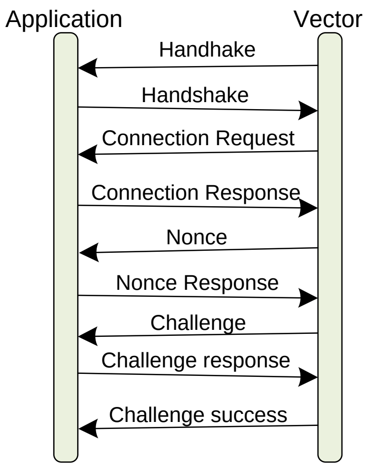


Figure 30: *Sequence for initiating communication with Vector*

1. The application opens Bluetooth LE connection (retrieving the service and characteristics handles), and subscribes to the "read" characteristic (see Appendix E 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.

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

Table 13: *Parameters for Handshake message*

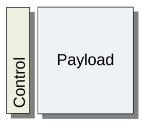
3. The application sends the handshake back
4. Then the Vector will send a *connection request,* consisting of the public key to use for the session. The application's response depends on whether this is a first time pairing, or a reuse.
    a. First time pairing requires that Vector have already been placed into pairing mode prior to connecting to Vector. The application keys should be created (see section *24.3.1 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 *24.3.1 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.

24.2. 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:



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 bytes remaining, repeat from step 2.

24.3. 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];
```

***Example 1:** Bluetooth LE encryption structures*

The variables mean:

***Table 14:** 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

24.3.1 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);
```

***Example 2:** Bluetooth LE key pair*

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);
```

***Example 3:** Bluetooth LE encryption & decryption keys*

24.3.2 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?}

24.3.3 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);
```

***Example 4:** Decrypting a Bluetooth LE message*

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);
```

***Example 5:** Encrypting a Bluetooth LE message*

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

24.4. 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:

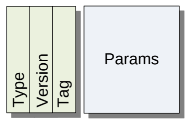


***Figure 31:** The format of an RTS frame*

- 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.

24.5. FETCHING A LOG

The process to set up a Bluetooth LE communication with Vector is complex. The sequence has many steps:

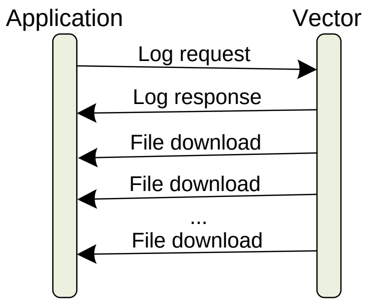


Figure 32: *Sequence for initiating communication with Vector*

The log request is sent to Vector. In principal this includes a list of the kinds of logs (called filter names) to be included. In practice, the "filter name" makes no difference.

Vector response, and if there will be a file sent, includes an affirmative and a 32-bit file identifier used for the file transfer.

Vector zips the log files up (as a tar.bz2 compressed archive) and sends the chunks to the application. Each chunk has this file identifier. (Conceptually there could be several files in transfer at a time.)

The file transfer is complete when the packet number matches the packet total.

25. 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 15: *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		$1a_{16}$	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

25.1. APPLICATION CONNECTION ID

?

25.1.1 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 16: *Parameters for Application Connection Id request*

25.1.2 Response

There is no response.

25.2. CANCEL PAIRING

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

25.2.1 Request

The command has no parameters.

25.2.2 Response

There is no response.

25.3. CHALLENGE

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

25.3.1 Request

The parameters of the request body are:

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

Table 17: *Parameters for challenge request*

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

25.3.2 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 18: *Parameters for challenge response*

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

25.4. CHALLENGE SUCCESS

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

25.4.1 Request

The command has no parameters.

25.4.2 Response

There is no response.

25.5. CLOUD SESSION

This command is used to request a cloud session.

25.5.1 Command

The parameters of the request body are:

Offset	Size	Type	Parameter	Description
0	2	uint16_t	*session token length*	The number of bytes in the session token; may be 0
2	varies	uint8_t	*session token*	The session token, as received from the cloud server.[26]
	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. Informational only. The mobile app uses the name of the mobile device. 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. Informational only. The mobile uses "companion-app". version >= 5

***Table 19:** Parameters for Cloud Session request*

25.5.2 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 21: Cloud status enumeration*
2	1	uint16_t	*client token GUID length*	The number of bytes in the client token GUID; may be 0
	varies	uint8_t[]	*client token GUID*	The client token GUID. The client token GUID should be saved for future use.

***Table 20:** 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

***Table 21:** Cloud status enumeration*

[26] https://groups.google.com/forum/#!msg/anki-vector-rooting/YlYQsX08OD4/fvkAOZ91CgAJ
https://groups.google.com/forum/#!msg/anki-vector-rooting/XAaBE6e94ek/OdES50PaBQAJ

5	authorized as secondary
6	reauthorization

25.6. CONNECT

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

25.6.1 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 22: *Parameters for Connection request*

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

25.6.2 Response

The parameters for the connection response message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*connectionType*	See *Table 24: Connection types enumeration*
1	32	uint8_t[32]	*publicKey*	The public key to use for the connection

Table 23: *Parameters for Connection Response*

The connection types are:

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

Table 24: *Connection types enumeration*

The application sends the response, with its publicKey (see section *24.3 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.

25.7. 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.

25.7.1 Request

The command has no parameters.

25.7.2 Response

There is no response.

25.8. FILE DOWNLOAD

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

25.8.1 Request

There is no direct request.

25.8.2 Response

The parameters of the response body are:

Table 25: *Parameters for File Download request*

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	uint16_t	*length*	The number of bytes to follow (can be 0)
	varies	uint8_t[length]	*bytes*	The bytes of this file chunk

25.9. LOG

This command is used to request the Vector send a compressed archive of the logs.

25.9.1 Request

The parameters of the request body are:

Table 26: *Parameters for Log request*

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

Each filter entry has the following structure:

Table 27: *Log filter*

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

25.9.2 Response

It can take several seconds for Vector to prepare the log archive file and send a response. The response will be a "log response" (below) and a series of "file download" responses.

The parameters for the response message:

Table 28: *Parameters for Log Response*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*exit code*	
1	4	uint32_t	*file id*	A 32-bit identifier that will be used in the file download messages.

25.10. NONCE

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

25.10.1 Request

The parameters for the nonce request message:

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

Table 29: *Parameters for Nonce request*

25.10.2 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

Table 30: *Parameters for Nonce response*

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

25.11. OTA UPDATE

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

25.11.1 Request

The parameters of the request body are:

Table 31: *Parameters for OTA request*

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

25.11.2 Response

The response will be one or more "OTA response" indicating the status of the update, or errors. Status codes >= 200 indicate that the update process has completed. The update has completed the download when the current number of bytes match the expected number of bytes.

The parameters for the response message:

Table 32: *Parameters for OTA Response*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*status*	See *Table 33: 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

The OTA status codes are:

Table 33: *OTA status enumeration*

Status	Meaning
0	idle
1	unknown
2	in progress
3	complete
4	rebooting
5	error
200…	Status codes from the update-engine. See Appendix C, *Table 63: OTA update-engine status codes*.

Note: the status codes 200 and above are from the update-engine, and are given in Appendix C.

25.12. RESPONSE

This message will be sent on the event of an error. Primarily if the session is not cloud authorized and the command requires it.

Table 34: *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*	The number of bytes in the string that follows.
	varies	uint8_t [length]	*text*	A text error message.

25.13. SDK PROXY

This command is used to pass the gRPC/protobufs messages to Vector over Bluetooth LE. It effectively wraps a HTTP request/response. Note: the HTTPS TLS certificate is not employed with this command.

25.13.1 Request

The parameters of the request body are:

Table 35: *Parameters for the SDK proxy request*

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)

25.13.2 Response

The parameters for the response message:

Table 36: *Parameters for the SDK proxy Response*

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)

25.14. SSH

This command is used to request the Vector allow SSH. It is reported that only the developer releases support SSH; it is not known which versions are applicable. It does not appear that SSH can be enabled in the release firmware.

25.14.1 Request

The parameters for the request message:

***Table 37:** 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 38:** 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

25.14.2 Response

The response has no parameters.

25.15. STATUS

This command is used to request basic info from Vector.

25.15.1 Request

The request has no parameters.

25.15.2 Response

The parameters for the response message:

Table 39: *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 40: 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*	0 if the Bluetooth
	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 40: *WiFi state enumeration*

Index	Meaning
0	Unknown
1	Online
2	Connected
3	Disconnected

25.16. WIFI ACCESS POINT

This command is used to request that the Vector act as a WiFi access point. This command requires that a "cloud session" have been successfully started first (see section *25.5 Cloud session*).

If successful, Vector will provide a WiFi Access Point with an SSID that matches his robot name.

25.16.1 Request

The parameters of the request body are:

Table 41: *Parameters for WiFi Access Point request*

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

25.16.2 Response

If the Bluetooth LE session is not cloud authorized a "response" message will be sent with this error. Otherwise the WiFi Access Point response message will be sent.

The parameters for the response message:

Table 42: *Parameters for WiFi Access Point Response*

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

25.17. WIFI CONNECT

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

25.17.1 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 44: WiFi authentication types enumeration*
	1	uint8_t	*hidden*	0 the access point is not hidden; 1 it is hidden

***Table 43:** 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 44:** WiFi authentication types enumeration*

25.17.2 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 40: WiFi state enumeration*
	1	uint8_t	*connect result*	version >= 3

***Table 45:** Parameters for WiFi Connect command*

25.18. WIFI FORGET

This command is used to request Vector to forget a WiFi SSID.

25.18.1 Request

The parameters for the request message:

Table 46: *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

25.18.2 Response

The parameters for the response message:

Table 47: *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

25.19. WIFI IP ADDRESS

This command is used to request Vector’s WiFi IP address.

25.19.1 Request

The request has no parameters

25.19.2 Response

The parameters for the response message:

Table 48: *Parameters for WiFi IP Address response*

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

25.20. WIFI SCAN

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

25.20.1 Request

The command has no parameters.

25.20.2 Response

The parameters for the response message:

Table 49: *Parameters for WiFi scan response*

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

Each access point has the following structure:

Table 50: *Parameters access point structure*

Offset	Size	Type	Parameter	Description
0	1	uint8_t	*auth type*	The type of authentication to employ; see *Table 44: 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

CHAPTER 12

The Cloud Services

This chapter is about the remote servers that provide functionality for Vector.

- JSON document storage server
- The crash uploader
- The diagnostic logger
- The token/certificate system
- The natural language processing

26. CONFIGURATION

The server URLs are specified in "`/anki/data/assets/cozmo_resources/config/server_config.json`" (The path to this JSON file is hardcoded in the application binaries.)

***Table 51:** The cloud services configuration file*

Element	Description & Notes
appkey	A base64 token used to communicate with servers. "oDoa0quieSeir6goowai7f"
check	The server to use for connection checks
chipper	The natural language processing server
jdocs	The remote JSON storage server
logfiles	The server to upload log files
tms	The token server where Vector gets authentication items like certificates and tokens

The crash upload URL is given in `/anki/etc/vic-crashuploader.env`

The OTA download URL is given in `/anki/etc/update-engine.env`

The DAS server to contact is given in `/anki/data/assets/cozmo_resources/config/DASConfig.json` (This path is hardcoded in vic-DASMgr)

27. JDOCS SERVER

The Vic-Cloud services stores information on a "JDocs" server. This unusual name appears to be short for "JSON Documents."

Vic-Cloud uses the "`jdocs`" tag in the cloud services configuration file to know which server to contact. It uses the file "`anki/config/engine/jdocs_config.json`" to adjust how ofteni t contacts the server. (The path to these json files are hardcoded in Vic-Cloud.)

The interactions are basic: store, read, and delete a JSON blob by an identifier. The description below[27] gives the JSON keys, value format. It is implemented as gRPC/protobufs interaction over HTTP.

27.1. JDOCS INTERACTION

Jdoc:

- doc_version
- fmt_version
- client_metadata
- json_doc

27.2. DELETE DOCUMENT

27.2.1 Request

DeleteDocReq

- account
- thing – the thing id is a ‘vic:’ followed by the serial number
- doc_name

27.2.2 Response

DeleteDocResp

27.3. ECHO TEST

27.3.1 Request

EchoReq

- data

27.3.2 Response

EchoResp

- data

27.4. READ DOCUMENTS

27.4.1 Request

ReadDocsReq

- account
- thing
- items

[27] The protocol was specified in Google Protobufs. Vic-Cloud and Vic-Gateway were both written in Go. There is enough information in those binaries to reconstruct significant portions of the Protobufs specification in the future.

27.4.2 Response

ReadDocsResp

- items

27.5. READ DOCUMENT ITEM

27.5.1 Request

ReadDocsReq_Item

- doc_name
- my_doc_version

27.5.2 Response

ReadDocsResp_Item

- status (int)
- doc

27.6. WRITE DOCUMENT

27.6.1 Request

WriteDocReq

- account
- thing
- doc_name
- doc

27.6.2 Response

WriteDocResp

- status (int)
- latest_doc_version

28. NATURAL LANGUAGE PROCESSING

The “knowledge graph” Q&A server is done by Sound Hound.

PART IV

Maintenance

- SETTINGS, PREFERENCES AND STATISTICS.
- SOFTWARE UPDATES. How Vector's software updates are applied.

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

CHAPTER 15

Settings, Preferences, Statistics

This chapter is about:

- The owner's account settings and entitlements
- The robot's settings (owner preferences)
- The robot's lifetime stats

29. THE ARCHITECTURE

The architecture for setting and storing settings, statistics, account information is:

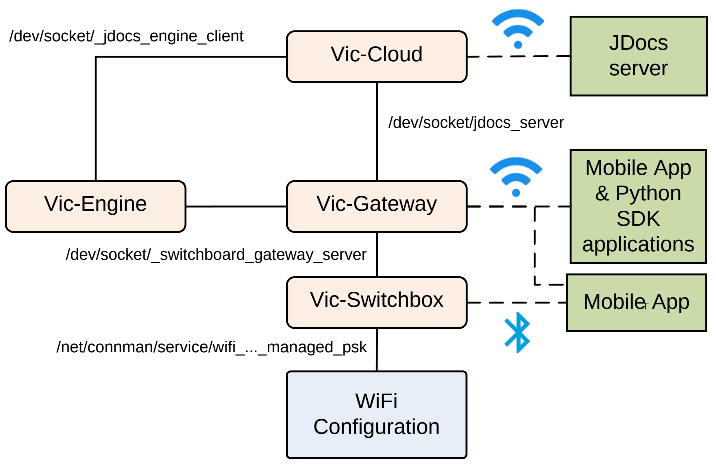

/dev/socket/_jdocs_engine_client
Vic-Cloud
JDocs server
/dev/socket/jdocs_server
Mobile App & Python SDK applications
Vic-Engine
Vic-Gateway
/dev/socket/_switchboard_gateway_server
Vic-Switchbox
Mobile App
/net/connman/service/wifi_..._managed_psk
WiFi Configuration

Figure 33: *The architecture for storing preferences, account info, entitlements, and tracking stats*

The Vic-Cloud service accesses information on a remote server.

The Vic-Switchbox interacts with the WiFi subsystem (connman) to allow the mobile App to set the preferred WiFI network to use. The mobile app must use Bluetooth LE to do this.

Vic-Gateway interacts with the mobile App and SDK programs to changes the robot settings.

Vic-Engine receives the preferences from the Vic-Cloud and Vic-Gateway, to carry out an changes in behaviour of Vector.

30. WIFI CONFIGURATION

The WiFi configuration (aka settings or preferences) is entirely local to the Vector robot. The information about the WiFi settings is not stored remotely.

The mobile application can configuration the WiFi settings via Vic-Switchbox commands. The WiFi is managed by connman thru the Vic-Switchbox:

- To provide a list of WiFi SSIDs to the mobile app
- To allow the mobile app to select an SSID and provide a password to
- Tell it forget an SSID
- To place the WiFi into Access Point mode

31. THE OWNER ACCOUNT INFORMATION

The owner account information is sent from the mobile application to Anki servers at time of registration and setting up a Vector. The owner account information includes:[28]

Table 52: *The owners account information*

JSON Key	units	Description & Notes
user_id	base64	A base64 token to identify the user
created_by_app_name	string	The name of the mobile application that register the owner. Example: "chewie"
created_by_app_platform	string	The mobile OS version string when the mobile application created the owners account. Example "ios 12.1.2; iPhone8,1
created_by_app_version	string	The version of the mobile application that register the owner. Example: "1.3.1"
deactivation_reason		
dob	YYYY-MM-DD	The owner's date of birth (the one given at time of registration)
drive_guest_id	GUID	A GUID to identify the owner. This is the same as the "player_id"
email	string	The email address used to register the account; the same as the user name.
email_failure_code		The reason that the email was unable to be verified
email_is_blocked	boolean	
email_is_verified	boolean	True if the email verification has successfully completed. False otherwise.
email_lang	IETF language tag	The IETF language tag of the owners language preference. example: "en-US"
family_name	string	The surname of the owner; null if not set
gender	string	The gender of the owner; null if not set
given_name	string	The given of the owner; null if not set
is_email_account	boolean	
no_autodelete	boolean	
password_is_complex	boolean	
player_id	GUID	A GUID to identify the owner. This is the same as the "drive_guest_id"

[28] It is not clear why there is so much information, and why this is sent from JDocs in so many cases.

purge_reason		
status	string	Example “active”
time_created	string	The time, in ISO8601 format, that the account was created
user_id	base64	A base64 token to identify the owner
username	string	Same as the email address

32. PREFERENCES & ROBOT SETTINGS

The following settings & preferences are stored in (and retrieved from) the JDoc server:

***Table 53:** The robot settings.*

Key	units	Description & Notes
button_wakeword	?	
clock_24_hour	boolean	
default_location		
dist_is_metric	boolean	
eye_color	enumeration	The colour used for the eyes. The colour is referred to by one of an enumerated set. (Within the SDK, the eyes can be set to a colour by hue and saturation, but this is not permanent.)
locale		American English, UK English, Australian English, German, French, Japanese, etc.
master_volume		
temp_is_fahrenheit	boolean	
time_zone		

33. OWNER ENTITLEMENTS

An entitlement is a family of features or resources that the program or owner is allowed to use. It is represented as another set of key-value pairs.

The only entitlements that Vector supports are “kickstarter eyes”

34. ROBOT LIFETIME STATS

The following lifetime statics 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.

***Table 54:** The robot lifetime stats schema*

Key	units	Description & Notes
Alive.seconds	seconds	Vector’s 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		
Odom.LWheel		The left wheel odometer
Odom.Rwheel		The right wheel odometer

Odom.Body		
Pet.ms	ms	The number of milliseconds petted?

CHAPTER 16

The Software Update process

This chapter is about the software update process

- The software architecture
- The firmware update process
- How to extract official program files

35. THE ARCHITECTURE

The architecture for updating Vector's firmware is:

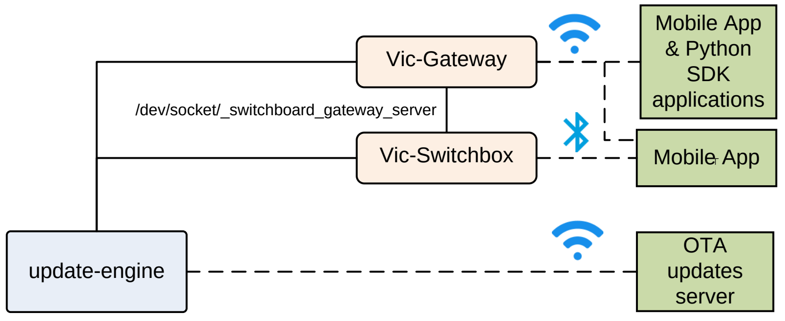

Vic-Gateway
/dev/socket/_switchboard_gateway_server
Vic-Switchbox
update-engine
Mobile App & Python SDK applications
Mobile App
OTA updates server

***Figure 34:** The architecture for updating Vector's software*

The Vic-Gateway and Vic-Switchbox both may interact with the mobile App and SDK programs to receive software update commands, and to provide update status information.

The update-engine is responsible for downloading the update, validating it, applying it, and providing status information to Vic-Gateway and Vic-switchbox. [It isn't know yet how they kick off the update]. The update-engine provides status information in a set of files with the "`/run/update-engine`" folder

36. THE UPDATE FILE

The update files are TAR files with a suffix "OTA". The TAR file has a fixed structure, with some of the files encrypted. There are 3 kinds of update files

- Factory updates. These modify the `RECOVERY` and `RECOVERYFS` partitions.
- Production updates. These modify the `ABOOT`, `BOOT`, and `SYSTEM` partitions
- Delta updates

The archive contains 3 to 5 files, and they must be in a specific order:

1. manifest.ini
2. manifest.sha256
3. apq8009-robot-delta.bin.gz (optionally encrypted). This is present only in delta updates
4. apq8009-robot-emmc_appsboot.img.gz (optionally encrypted). This is present only in factory updates. It will be applied to the ABOOT partition.
5. apq8009-robot-boot.img.gz (optionally encrypted) . This is not present in delta updates. In factory updates it will be applied to the RECOVERY partition; otherwise it will be applied to the BOOT partition.
6. apq8009-robot-sysfs.img.gz (optionally encrypted). In factory updates it will be applied to the RECOVERYFS partition; otherwise it will be applied to the SYSTEM partition.

36.1. MANIFEST.INI

The manifest.ini is checked by verifying its signature[29] against manifest.sha256 using a secret key (/anki/etc/ota.pub):

```
openssl dgst \
  -sha256 \
  -verify /anki/etc/ota.pub \
  -signature /run/update-engine/manifest.sha256 \
  /run/update-engine/manifest.ini
```

***Example 6:** Checking the manifest.ini signature*

Note: the signature check that prevents turning off encryption checks in the manifest below. At this time the signing key is not known.

All forms of update have a [META] section. This section has the following structure:

***Table 55:** manifest.ini META section*

Key	Description
ankidev	0 if production release, 1 if development
manifest_version	Acceptable version include 0.9.2, 0.9.3, 0.9.4, 0.9.5, or 1.0.0
num_images	The number of img.gz files in the archive. The number must match that of the type of update file it is. 1, 2, or 3
qsn	The Qualcomm Serial Number; if there are three images (ABOOT, RECOVERY, RECOVERYFS) present, the firmware is treated as a factory update. The QSN must match the robot's. Optional.
reboot_after_install	0 or 1. 1 to reboot after installing.
update_version	The version that the system is being upgrade to, e.g. 1.6.0.3331

After the [META] section, there are 1 to 3 sections, depending on the type of update:

- A delta update has a [DELTA] section
- A regular update has a [BOOT], [SYSTEM] sections; both must be present/
- A factory update has [ABOOT], [RECOVERY], and [RECOVERYFS] sections; all 3 must be present.

[29] I'm using the information originally at: https://github.com/GooeyChickenman/victor/tree/master/firmware

Each of these sections has the same structure:

***Table 56:** manifest.ini image stream sections*

Key	Description
base_version	The version that Vector must be at in order to accept this update. Honored only in delta updates.
bytes	The number of bytes in the uncompressed archive
compression	gz (for gzipped). This is the only supported compression type.
delta	1 if this is a delta update; 0 otherwise
encryption	1 if the archive file is encrypted; 0 if the archive file is not encrypted.
sha256	The digest of the decompressed file must match this
wbits	31. Not used buy update-engine

36.2. HOW TO DECRYPT THE OTA UPDATE ARCHIVE FILES[30]

How to decrypt the OTA update archive files:

```
openssl enc -d -aes-256-ctr -pass file:ota.pas -in apq8009-robot-boot.img.gz -out
apq8009-robot-boot.img.dec.gz
openssl enc -d -aes-256-ctr -pass file:ota.pas -in apq8009-robot-sysfs.img.gz -out
apq8009-robot-sysfs.img.dec.gz
```

***Example 7:** Decrypting the OTA update archives*

To use OpenSSL 1.1.0 or later, add "-md md5" to the command:

```
openssl enc -d -aes-256-ctr -pass file:ota.pas -md md5 -in apq8009-robot-boot.img.gz -
out apq8009-robot-boot.img.dec.gz
openssl enc -d -aes-256-ctr -pass file:ota.pas -md md5 -in apq8009-robot-sysfs.img.gz -
out apq8009-robot-sysfs.img.dec.gz
```

***Example 8:** Decrypting the OTA update archives with Open SSL 1.1.0 and later*

Encrypting the files is a whole other kettle of fish

37. THE UPDATE PROCESS

37.1. STATUS DIRECTORY

The update-engine provides its status thru a set of files in the `/run/update-engine` folder.

***Table 57:** update-engine status file*

File	Description
done	If this file exists, the update has completed
error	The error code representing why the update failed. See Appendix C, *Table 63: OTA update-engine status codes.*
expected-download-size	The expected file size (the given total size of the OTA file) to download.
expected-size	In non-delta updates, the total number of bytes of the unencrypted image files. This is the sum of the "bytes" field in the sections.
progress	Indicates how many of the bytes to download have been completed,

[30] https://groups.google.com/forum/#!searchin/anki-vector-rooting/ota.pas%7Csort:date/anki-vector-rooting/YlYQsX08OD4/fvkAOZ91CgAJ

	or how much of the partitions have been written.

This folder also holds the unencrypted, uncompressed files from the OTA file:

- `manifest.ini`
- `manifest.sha256`
- `delta.bin`
- `aboot.img`
- `boot.img`

37.2. PROCESS

The update process if there is an error at any step, skips the rest, deletes the bin and img files.

1. Remove everything in the status folder
2. Being downloading the OTA file. It does *not* download the TAR then unpack it. The file is unpacked as it is received.
3. Copies the manifest.ini to a file in the status folder
4. Copies the manifest.sha256 to a file
5. Verifies the signature of the manifest file
6. Validates that the update to the OTA version is allowed. This includes a check that it is to a newer version number, and the developer vs production firmware type matches whether this Vector is a developer or production.
7. If this is factory update, it checks that the QSN in the manifest matches Vector's QSN.
8. It marks the target partition slots as unbootable
9. Checks the img and bin contents
    a) delta
    b) boot & system
    c) If this is a factory update, it aboot, recovery, recoveryfs
10. If this is a factory update:
    a) Sets /run/wipe-data. This will trigger erasing all of the user data on the next startup
    b) Makes both a and b slots for BOOT and SYSTEM partitions as unbootable
11. If this is not factory update
    a) Sets the new target slot as active
12. Deletes any error file
13. Sets the done file
14. Posts a DAS event robot.ota_download_end to success + next version
15. If the reboot_after_install option was set, reboots the system

38. RESOURCES

https://source.android.com/devices/bootloader/flashing-updating
Describes the a/b process as it applies to android

References & Resources

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

39. CREDITS

Credit and thanks to Anki, C0RE, Melanie T for access to the file-systems, decode keys, and board shots. Fictiv for board shots. The board shots that help identify parts on the board and inter-connection on the board. HSReina for Bluetooth LE protocol information.

40. REFERENCE DOCUMENTATION AND RESOURCES

40.1. ANKI

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

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

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

Stein, Andrew; *Making Cozmo See*, Embedded Vision Alliance, 2017 May 25
https://www.slideshare.net/embeddedvision/making-cozmo-see-a-presentation-from-anki
https://youtu.be/Ypz7sNgSzyI

40.2. OTHER

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

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

Sriram, Swetha, *Anki Vector Robot Teardown*, Fictiv, 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/

cozmopedia.org

PyCozmo
https://github.com/zayfod/pycozmo/tree/master/pycozmo

40.3. QUALCOMM

Although detailed documentation isn't available for the Qualcomm APQ8009, there is documentation available for the sibling APQ8016 processor.

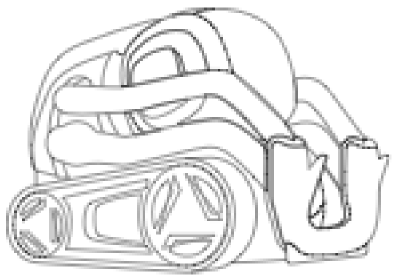
Qualcomm, *APQ8016E Application Processor Tools & Resources*, https://developer.qualcomm.com/hardware/apq-8016e/tools

Qualcomm, *DragonBoard™ 410c based on Qualcomm® Snapdragon™ 410E processor ADB Debugging Commands Guide*, LM80-P0436-11, Rev C, 2016 Sep lm80-p0436-11_adb_commands.pdf

Qualcomm, *DragonBoard™ 410c based on Qualcomm® Snapdragon™ 410E processor Software Build and Installation Guide, Linux Android*, LM80-P0436-2, Rev J, 2016 Dec lm80-p0436-2_sw-build-and-install_gd_linux_android_dec2016.pdf

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
- FAULT AND STATUS CODES. This appendix provides describes the system fault codes, and update status codes.
- FILE SYSTEM. This appendix lists the key files that are baked into the system.
- 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.
- PLEO. This appendix gives a brief overview of the Pleo animatronic dinosaur, an antecedent with many similarities.



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

APPENDIX A

Abbreviations, Acronyms, Glossary

Table 58: *Common acronyms and abbreviations*

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	*unknown (diagnostic/data analytics service?)*
DFU	device firmware upgrade
EEPROM	electrical-erasable programmable read-only memory
EMR	*unknown (emergency mode recovery?)*
ESD	electro-static discharge
ESN	electronic serial number
FBS	flat buffers
FDE	full disc encryption
GPIO	general purpose IO
GUID	globally unique identifier (effectively same as UUID)
I2C	inter-IC communication
IMU	inertial measurement unit
IR	infrared
JDocs	JSON Documents
JSON	javascript object notion
JTAG	Joint Test Action Group
LCD	liquid crystal display
LED	light emitting diode
LUKS	linux unified key setup
MCU	microcontroller
mDNS	multicast domain name service (DNS)

MEMS	micro-electromechanical systems
MISO	master-in, slave-out
MOSI	master-out, slave-in
MPU	microprocessor
MSRP	manufacturer's suggest retail price
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
PWM	pulse width modulation
QSN	Qualcomm serial number
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 (effectively same as GUID)
vic	short Victor (Vector's working name)

Table 59: *Glossary of common terms and phrases*

Phrase	**Description**
A*	A path finding algorithm
aboot	The Android boot-loader used to launch Vector's linux system.
attitude	orientation
boot loader	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	
certificate	Vector generates an SSL certificate that can be used for the secure

	communications.
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.
client token	A string token provided by Vector that is passed with each SDK command.
control	motors and forces to move where and how it is told to. (smooth arcs)
D*-lite	A path-finding algorithm
device mapper verity (dm-verity)	A feature of the Linux kernel that checks the boot and RAM file systems for alteration, using signed keys
flash	A type of persistent (non-volatile) storage media.
guidance	Builds the desired path
navigation	Knowing where it is in the map
nonce	An initially random number, incremented after each use .
path planning	smooth arcs and line segments
pose	position and orientation of an object relative to a coordinate system
power source	Where the electric energy used to power Vector comes from.
rapidly-expanding random tree	A path-finding algorithm
recovery mode	
robot name	Vector robot name looks like “Vector-E5S6”. It is “Vector-“ followed by a 4 letters and numbers.
session token	A string token provided by the Anki servers that is passed to Vector to authenticate with him and create a *client token.*
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.
Trust Zone	A security mode on ARM processor where privileged/special code is run. This includes access to encryption/decryption keys.
universally unique identifier (UUID)	A 128bit number that is unique. (effectively same as GUID)

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 60: *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[31]	Used to craft the sounds https://www.audiokinetic.com/products/wwise/
Backtrace.io	A service that receives uploaded minidumps from applications in the field, and provides tools to analyze them. https://backtrace.io
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

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

	https://01.org/connman
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 Vector's internal software.
Google Breakpad	Google Breakpad is used to generate tracebacks and mini-dump files of programs that crash. Results are sent to htttp://backtrace.io https://chromium.googlesource.com/breakpad/breakpad
Google FlatBuffers	Google FlatBuffers is used to encode the animation data structures https://github.com/google/flatbuffers
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[32]	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; 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.

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

	https://developers.google.com/protocol-buffers
Pretty Fast FFT pffft	Julien Pommier's FFT implementation for single precision, 1D signals https://bitbucket.org/jpommier/pffft
Pryon, Inc	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
Qualcomm TBD	Qualcomm's device drivers and other kit appear 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, Inc	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.
Signal Essence	Designed the microphone array, and the low-level signal processing of audio input. https://signalessence.com/
Sound Hound, inc	Vector's Q&A "knowledge graph" is done by Sound Hound https://blog.soundhound.com/hey-vector-i-have-a-question- 3c174ef226fb
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 61:** Tools that can be used to analyze and patch Vector*

Tool	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

Fault and status codes

The following are system status codes that may be produced during startup:

Table 62: *The system fault codes*

Code	Meaning
1..10	Systemd failed…?
200…	Software update status code, see table below
700-705	Internal sensor out of range or failed
800	Vic-anim was unable to start or crashed
801	?
898	? “general hardware disconnect”
899	?
913	Vic-switchboard was unable to start or crashed
914	Vic-engine was unable to start or crashed
915	?
916	Vic-robot was unable to start or crashed
917	?
920	Vic-gateway-cert was unable to generate a x509 certificate for Vic-gateway
921	Vic-gateway was unable to start or crashed
923	Vic-cloud was unable to start or crashed
980	?

The following are the update-engine status codes that may be produced during the update process:

Table 63: *OTA update-engine status codes*

Status	Meaning
200	The TAR contents did not follow the expected order.
201	Unhandled section format for expansion, or The manifest version is not supported, or The OTA has the wrong number of images for the type, or The OTA is missing a BOOT or SYSTEM image, or The manifest configuration is not understood
202	Could not mark target, a, or b slot unbootable, or Could not set target slot as active
203	Unable to construct automatic update URL, or The URL could not be opened
204	The file wasn’t a valid TAR file, or is corrupt
205	The compression scheme is not supported, or Decompression failed, the file may be corrupt
207	Delta payload error

208	Couldn't sync OS images to disk, or Disk error while transferring OTA file.
209	The manifest failed signature validation; or the aboot, boot image, system image, or delta.bin hash doesn't match signed manifest
210	The encryption scheme is not supported.
211	Vector’s current version doesn’t match the baseline for a delta update.
212	The decompression engine had an unexpected, undefined error.
213	QSN doesn't match manifest
214	There is a mismatch: development Vectors can’t install release OTA firmware, and release Vectors can’t install development OTA firmware.
215	OTA transfer failed, due to timeout.
216	OS version name in the update file doesn’t follow an acceptable pattern, or it is not allowed to upgrade or downgrade from the current version to the new version.
219	Other unexpected, undefined error while transferring OTA file.

APPENDIX D

File system

This Appendix describes file systems

As the Vector uses the Android bootloader, it reuses – or at least reserves – many of the Android partitions[33] 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.

***Table 64:** The file system mount table*

Mount point	Partition name	Description & Notes
/	BOOT_A	The primary linux kernel and initramfs
/data[34]	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

The partition table[35] found on the Vector:

***Table 65:** The partition table*

Partition name	Size	Description & Notes
ABOOT	1 MB	The application boot loader, which may load the kernel, recovery, or fastboot.
ABOOTBAK	1 MB	The application boot loader, which may load the kernel, recovery, or fastboot.
BOOT_A	32 MB	These are the primary and backup linux kernel and initramfs. Updates modify the non-active partition, and then swap which one is active.
BOOT_B	32 MB	
CONFIG	512 KB	Configuration of TBD.
DDR	32 KB	Configuration of the DDR RAM.

[33] https://forum.xda-developers.com/android/general/info-android-device-partitions-basic-t3586565
[34] This is mounted by "mount-data.service' The file has a lot of information on how it unbricks
[35] Much information from: https://source.android.com/devices/bootloader/partitions-images

DEVINFO	1 MB	“device information including: is_unlocked (aboot), is_tampered, is_verified, charger_screen_enabled, display_panel, bootloader_version, radio_version etc. Contents of this partition are displayed by “fastboot oem device-info” command in human readable format. Before loading boot.img or recovery.img, [the] boot loader verifies the locked state from this partition.”
EMR	16 MB	Holds Vector’s Model, Serial Number, and such
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 boot loader 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 and backup system applications and libraries with application specific code. Updates modify the non-active partition, and then swap which one is active.
SYSTEM_B	896MB	
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 66: 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/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[36] */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[37]	Sets the fake time to the time file (Vector doesn't have a clock)
/run/fault_code	This is set to the fault code (see Appendix C) if a program is unable to carry out a significant task, or crashes. The fault display program may present this code on the LCD display.
/tmp/data_cleared	
/tmp/vision/neural_nets	

Key named device files employed in Vector binaries:

Table 67: 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	Serial connection with the base-board
/dev/ttyHSL0	Console log
/sys/class/gpio/gpio83	Used to control the camera power

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

APPENDIX E

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.

41. CUBE SERVICES

times and other feature parameters:

Table 68: *The Bluetooth LE services*

Service	UUID[38]	Description & Notes
Device Info Service[39]	$180A_{16}$	Provides device and unit specific info –it's manufacturer, model number, hardware and firmware versions
Generic Access Profile[40]	1800_{16}	The device name, and preferred connection parameters
Generic Attribute Transport[41]	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

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)

Table 69: *The Cube's Device info settings*

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

[38] All values are a little endian, per the Bluetooth 4.0 GATT specification
[39] http://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.device_information.xml
[40] http://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.generic_access.xml
[41] http://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.generic_attribute.xml

41.1. CUBE'S ACCELEROMETER SERVICE

Values are little-endian, except where otherwise stated.

Table 70: Cube's accelerometer service characteristics

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 accelerometer	
		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*
9590BA9C-5140-92B5-1844-5F9D681557A4$_{16}$	Write		Unknown

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

42. VECTOR SERVICES SERVICE

times and other feature parameters:

Table 71: Vector's Bluetooth LE services

Service	UUID[42]	Description & Notes
Generic Access Profile	1800$_{16}$	The device name, and preferred connection parameters
Generic Attribute Transport	1801$_{16}$	Provides access to the services.
Vector's Serial Service	FEE3$_{16}$	The service with which we can talk to Vector.

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

Table 72: The Vector's Device info settings

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

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

42.1. VECTOR'S SERIAL SERVICE

Table 73: *Vector's serial service characteristics*

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

APPENDIX F

Servers & Data Schema

This Appendix describes the servers that Vector contacts[43]

***Table 74:** The servers that Vector contacts.*

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.[44]
https://anki.sp.backtrace.io:6098	Vector posts crashes (linux minidumps) 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?

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

APPENDIX H

Phrases and their Intent

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

Table 75: *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
{same as quiet down}	Turn off
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

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?

Table 76: *The Vector questions phrases*

Some of them are internal strings, some are hardcoded values

APPENDIX J

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."[45]

42.2. 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[46]

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

42.3. 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

[45] https://news.ycombinator.com/item?id=17755596
[46] 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