Redundancy Overview

Redundant systems and components guard against any single hardware or software failure. Redundancy relies on the probability that two failures within a short duration are highly improbable. With critical systems, controller redundancy alone is insufficient. A fully redundant system requires end-to-end redundancy, system isolation to avoid the effects of cascading a fault, and failure detection of any critical component. This means all system components, including the network, must be redundant and actively monitored so the Mean Time To Repair (MTTR) of any single component is minimized. Failing to detect and repair even a fault in a backup component allows the system to effectively operate in a non-redundant mode (referred to as 'simplex' operation) until an inevitable failure of a primary component occurs, leaving the system with no fallback position to continue operating correctly.

A Redundant Logical Controller is typically made up of two or more physical controllers and defined by their Roll plus their State. The physical controllers are often designated the roles of Primary and Secondaries (or backups), where the Primary controller outputs are used to drive the system, and the Backup stands ready to take over as Primary in the event of a failure. By definition, the Primary is in the Active State, indicating it is operational and driving the system. Each of the Backup controllers can be in any one of the following states: Cold/Warm-Standby or Hot-standby. Hot-standby designates that the switchover of the Active role from Primary to Backup is fundamentally the time to detect the Primary has failed (that is, the Backup has all the state necessary to take over as Primary.) Cold and Warm-Standby may require a lag to gain sufficient state to take over as the Primary. This method employed is dependent upon the system redundancy switchover requirements.

Redundancy In a DDS Shipboard System

Shown in the figure blow is an example of a Redundant Shipboard Navigation System.

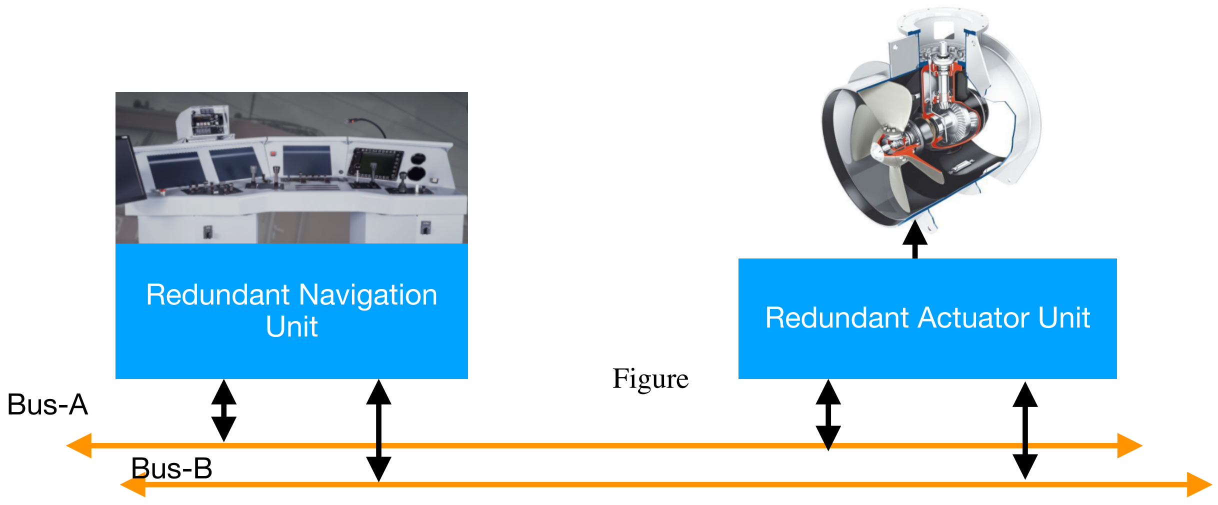

Figure 1.0 - Basic Architecture of an End-to-End Redundant Shipboard Navigation System

There are multiple approaches to providing redundancy. This document will present two redundancy architectures[1], specifically in the context of a ship's thrust/steerage control system using DDS. (1) A Duplex Redundancy design (see Figure 2 below) presenting a logical controller comprised of two controllers, each with two processors. The Primary Controller drives the system[2]. In the event that the two processors on either controller disagree, that controller would remove itself from the system and set an alarm (leaving the "other" controller to take over in the Active State). (2) a 3-way Voting design (see Figure 4 below) where the logical controller is comprised of three simplex controllers. Using Heartbeat messages and a common algorithm, such as the lowest BoardId and each board's state, the Active Controller is voted by two of the three controllers. Upon any controller failure, the two remaining controllers would, by algorithm, select the Active controller. The Active controller would be responsible for alarming and identifying any failed controller. The Heartbeat frequency or 'Deadline' would be the primary component of switchover time. At the cost of switchover time, the 3-way voting architecture has the advantages of 1) Using off-the-shelf hardware while ensuring a third arbiter to ensure a switchover to a capable controller, and 2) the controllers can be distributed and less prone to a catastrophic physical event such as an explosion or flooding. The Duplex Redundancy potentially has the capability, via specialized hardware, to allow detection and switchover within one system clock period.

Duplex Redundancy Detail

A Duplex Redundancy design, using specialized lock-step hardware, contains two statically designated controllers with no distinction of Role (Primary and Secondary) or state (Active and Standby). Each controller has two processors running identical[3] applications receiving and calculating the same information, sharing results between the two applications. On any one controller, if lockstep output data does not compare, specialized hardware on the controller disconnects the controller from the data bus (stops sending heartbeats). If a controller detects an application assert, process hang, or hardware failure, the application would assert a failure to remove itself from the system. Once the failed controller is restored, it will return to synchronous driving of outbound data.

Each controller would run a Network Alarm Participant (NAP) to independently monitor network heartbeats between controllers of each assigned bus to ensure a failure on either bus is alarmed for immediate repair (see Detect Network Failures - Alarm and Clear below) . Independent of alarming, the design might use Connext’s Interface Priority feature to change interfaces associated with each bus upon a failure.

[1] We won’t discuss so called n+1 redundancy where you have multiple controllers or devices active, perhaps load sharing, with one or more standby units in reserve.

[2] There is a form of hardware fault tolerance where neither controller is designated Primary or Secondary. Here both controllers drive the system simultaneously. Each controller has two processors running in cpu/bus clock lockstep. If any discrepancy is detected between the processors of a controller, hardware removes the controller from the bus within one clock cycle.

[3] To avoid identical software error resulting in erroneous agreement, the two applications could be written independently.

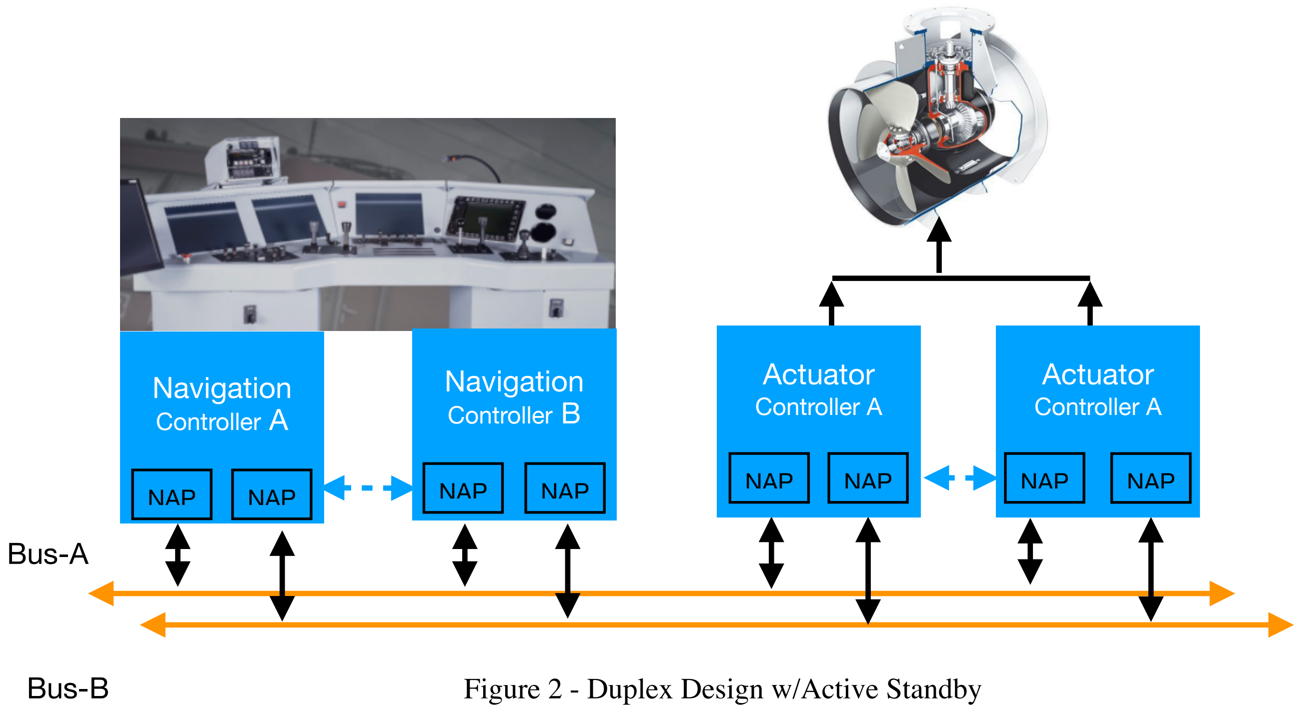

Figure 2 - Duplex Design w/Active Standby

Three-way voting Redundancy

With a Three-way-voting design, each logical controller unit is made up of three simplex controller components (denoted by the orange circle in the figure below). These can be off-the-shelf processing elements.

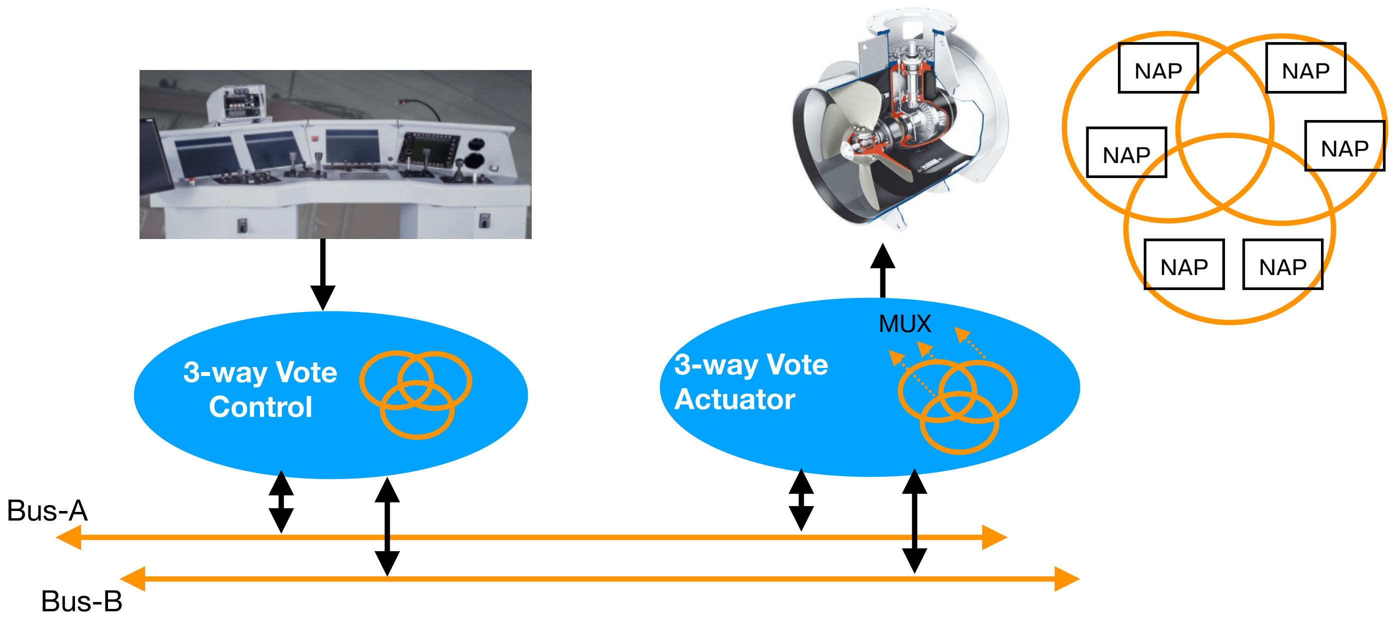

Figure 3 - Three-way Voting Architecture

Using heartbeat messages each controller can monitor the health of the other two controllers, using a voting algorithm to determine which of the operating controller would be the primary and which are backups. Like the duplex design, each of the like controller components would execute identical (see footnote 4 below) application processes and run a NAP on each interface to detect and alarm Network failure Independent of the NAP, the applications could use Connext Interface Priority to switch from one network to the other upon failure. Here too, the Active controller would send out a ControllerRole Topic as described in Active Controller Selection Process section above. It will alarm a failed controller for repair as described in Detect and Alarm and Clear a failed Controller above.

Redundancy Layer (RL) using DDS

To isolate and abstracts the application from a particular redundancy implementation we introduce a Redundancy Layer (RL). The RL is responsible for the following activities: 1) Active / Standby state selection, 2) failure detection and alarming, 3) switchover selection. A single API is presented to the application minimizing exposure of the redundancy implementation. In a distributed system, the RL is also be responsible for monitoring and alarming a network failure.

For a shipboard system, both the Redundant Navigation Unit Controller and the Redundant Actuator Controller could use the same Redundancy Layer.

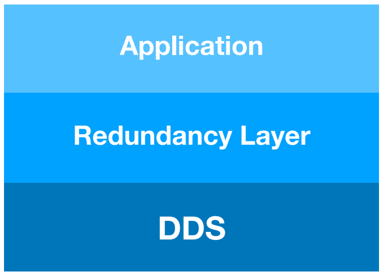

Figure 4 - Redundancy Abstraction Layer

Let's look at these responsibilities in detail in the context of our distributed shipboard system as shown in figure 1 above. Most of this discussion, with the exception of network monitoring applies only the the Three-way voting design, since the Duplex design is mostly hardware based and has no Active / Standby role distinction.

Active Controller Selection Process and Controller Alarms:

All like[4] redundant controllers (Navigation or Actuators) will initialize to the highest Standby state possible and send and receive some sort of Redundancy Layer Controller Heartbeat (RLCHB) between themselves. They will then enter the voting cycle as described in the next paragraph. The RLCHB is used to establish that the controller is alive and learn which controllers are capable to go to the Active state. The RLCHB topic should contain the source Controllers ID and its current state. The RLCHB is sent periodically with QoS of Best Effort, along with a deadline.

Each Active capable (meaning Hot-Standby) controller, after waiting two times the RLCHB time will send a VoteControllerState topic, listing its nomination, based on a common algorithm (e.g. lowest Controller ID) of potential Active, Standby1, and Standby2 (if applicable) state. Precedence should be given to any currently Active Controller to avoid unnecessary Active/Standby switching. Each capable controller, using its own vote and achieving two votes in agreement for a role, will take on that role[5], reflecting it in the next RLCHB it sends out. The VoteControllerState topic is sent with QoS of Transient Local Durability to allow late joining controller to send it's vote based

Each controller will reenter the voting cycle anytime it fails to receive RLCHB from any known controller to re-establish controller roles based on majority consensus. In the case of an Active Controller failure, this could occur either from an Active Controller failure, or a Standby controller communications link failure (network or onboard transceiver). With Redundant networks, if the failure was due to a communications link only one controller would attempt an updated vote which would be inconsistent

[4] One may want to also employ controller groups, in case there are multiple "like" logical controllers. That is beyond the scope of this document.

[5] This address the possibility of a controller (n) receiving heartbeats / topic but not able to send them. In this case the other two controller would be in agreement as seen by controller n.

with the other two controllers, alerting the Active Controller of the need to Alarm the system. In the case where the Active Controller's transceiver failed, and its RLCHBs are not received from either the standby, while the Active Controller will remain "Active" locally , it is isolated from the system and the Newly voted Active Controller will take precedence.

The Active controller will always alarm or clear any failed / recovered controller(s) Alarms.

Each controller "user data" (e.g. commands to navigate the ship) should set its exclusive ownership strength relative to its role (Active, Standby-One, and Standby-two in decreasing strength), so that only the Active controllers "user data" output will be seen by the system.

Detect Network Failures - Alarm and Clear

Connext DDS 7.2 offers multiple interface prioritization. Meaning, the user can specify two interfaces in priority order, in the event one fails, the other will take over automatically. This ensures that reliably sent messages are delivered exactly once, even through a switch over. There are two limitations with this solution that need to be addressed.

1. An interface is an IP address and not necessarily a different network. So the user has to nail up IP addresses, or ensure each interface corresponds to a different physical network.
2. While Connext detects failure of the active network (Primary), it does not continually check that the backup network has failed (perhaps, long ago).

To ensure the backup network is available, a Network Alarm Participant (NAP) (i.e. a separate Connext Application) would need to be run minimally twice (one for each network) between 'like' controllers, sending what we'll call Redundancy Layer Network Heartbeats (RLNHB), between them (Bidirectionally on each interface). Upon an appropriate DDS Deadline miss event, a Network Alarm would be raised. This would ensure the user is notified of a network failure.

The table below shows the logical conclusion based on Network Alarm Topic state.
Note: NAPs send data between only Like controllers on each respective network, i.e. NavCtrlrP on N1 would send to NavCtrlrS-N1 etc.

Nav Ctrlr P-N1	Nav Ctrlr P-N2	Nav Ctrlr S-N1	Nav Ctrlr S-N2	Actuator Ctrlr P-N1	Actuator Ctrlr P-N2	Actuator Ctrlr S-N1	Actuator Ctrlr S-N2	FAULT
Good	Good	Good	Good	Good	Good	Good	Good#	None
Good	Good	Good	Good	Good	Good	Good	Fail #**	Actuator Ctrl P-N2 Fail or Actuator Ctrl S-N2**
Good	Good	Good	Good	Fail	Fail	Fail	Fail	Either Actuator Ctrlr P or S Down ##
Fail	Fail	Fail	Fail	Good	Good	Good	Good	Either Nav Controller P or S Down##

Nav Ctrlr P-N1	Nav Ctrlr P-N2	Nav Ctrlr S-N1	Nav Ctrlr S-N2	Actuator Ctrlr P-N1	Actuator Ctrlr P-N2	Actuator Ctrlr S-N1	Actuator Ctrlr S-N2	FAULT
Fail	Good	Fail	Good	Fail	Good	Fail	Good	Network1 Failure
Good	Fail	Good	Fail	Good	Fail	Good	Fail	Network2 Failure
Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Double Fault - Both Networks Fail

* P-N1, S-N2 etc is the 'Like Controller' Primary/Secondary Role, and Network interface (N1 or N2).
# Good means RLNHB received, Fail means RLNHB not received on the interface
** Any Single failure indicates the Actuator either the driver on the sending controller failed or receiver on receiving controller failed (to keep the table simple, we won't expand all eight scenarios out)
## This condition is directly detected by the RLCHB so no alarm should be raised by the Network Alarm Participant(NAP).
### This table shows only two controllers of like types, and technically all that's needed to run NAPs on both networks. However to detect transceiver failures on any controller, NAPs should be run on all like controllers (i.e. all three in the case of three-way voting).

Each NAP will raise an alarm for any detected Failure, however , each NAP will subscribe to all Like Controller RLNHBs to actually conclude a network alarm or transceiver / Controller fault (Red Faults above) which is not detectable using RLCHBs. The NAP should not raise failed Controller Alarms (Yellow Faults) as those are handled directly by the controller using RLCHBs.

Topic	Qos	Type	Written by	Read by
RLCHB	Best effort,	Periodic	All RLs	All RLs
RLNHB	Best effort,	Periodic	All NAP applications	All NAP applications
Vote	Reliable	As Needed	All RLs	All RLs
Controller Alarm	Reliable	As Needed	Network Alarm Participant	Network Alarm Participant
Network Alarm	Reliable	As Needed	Network Alarm Participant	Network Alarm Participant

Summary of Reliability LayerNew Topics

Conclusion

While a 3-way voting redundancy has the increased cost of an additional controller as well as increased software complexity over a Duplex architecture, it meets the full requirements of 'no single point of failure', and can use Commercial-off-the-shelf (COTS) hardware.

Pixytracker App Demo - Three-way Voting

For Demonstration purposes, just the Pixytracker App portion of the Pixytracker system (shown below) will be made redundant using the Three-way Voting Architecture described above. This will require moving the tracker app from a Mac-based host to three Raspberry Pi 3B+ single-board computers. This demo WILL NOT run on duplicate networks ("The network is not the adversary.") or duplicate Pixycam apps.

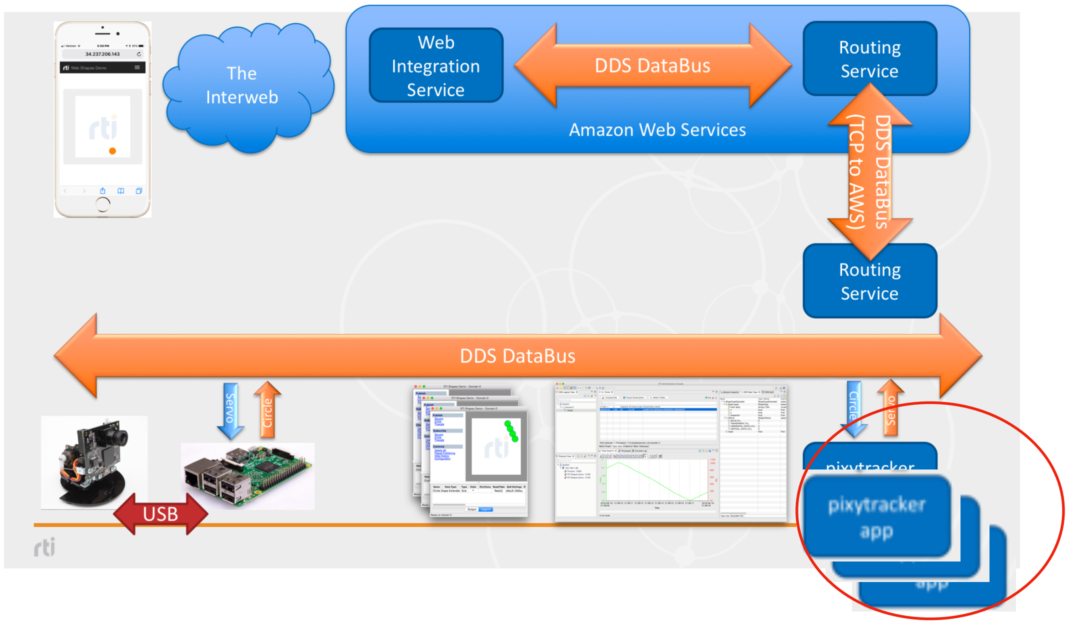

Figure 5 Pixytracker application w/3-way Voting logic

The demo will first bring up one Raspberry running the tracker app[6]. Upon rebooting the Raspberry, we can show no tracking takes place while the Raspberry is offline. We'll then bring up three Raspberrys showing that bouncing any one of the Raspberry's causes no perceivable change in pixy cam tracking. A Status LED will be connected to each Controller with the following status: Green LED = Active , Yellow LED = Standby (one or two), Red LED = Failed. Alarms will be subscribed to by an "Alarm Monitoring" process running on a host laptop.

Modification to PixyShapes.cxx (app on the Raspberry)

Because we are only protecting the Pixytracker app and not detecting network failure or pixy cam failure - only QoS settings to enable ownership strength on the pixy/servo_control topic need be modified (with Reliable Reliability with Deadline 200ms.). No code modifications should be required of the pixyShapes code.

[6] currently running on my Laptop

Modifications to PixyTracker.cxx to make it redundancy (3-way voting) capable

Existing Topics (and Modifications):

Circle (No changes)

Pixy/servo_control (Changes)
Data_reader: N
Data_writer: Y

Changes: QoS Only: Enable Ownership Strength (and Reliable Reliability with Deadline 100ms.) Each PixyTracker Controller will define a unique strength and ALWAYS publish using this strength. Ownership strength will be determined either via an XML variable or on the command line.

Redundancy Layer New Topic Definitions

New Topics:
RLCHB - best effort/periodic four times per second (Voting time < 3 seconds) - contains the senders participant *Keyed* GUID for algorithm comparison. This topic will have a deadline of 2x its send rate.

Vote topic - sent durably by each controller. Contains senders GUID (Keyed) along with its vote of who (GUIDS) is the primary controller (GUID) followed by vote of Controller Relative Number 1, 2, 3 (lowest GUID to highest).

Tracker Ordinal Number (1-3) is used locally to identify itself and the LED position (see below) as described below in LED Operation. A controllers Relative number can change, however, the actual Primary Controller will not change unless it fails (I.e. hysteresis so we don't oscillate between primary controllers).

Voting Algorithm -

The voting algorithm will select Primary, Secondary, and Tertiary controllers (Trackers). Each controller will set the Ownership Strength of the Pixy/Servo topic relative to its role as follows: Primary (30), Secondary (20), Tertiary (10).

Upon a controller initializing, a controller will come up in one of two scenarios: 1) As part of a new system coming online (i.e., no controllers were previously running, no rolls have been established, and 2) As a late Joiner to an existing system. For each of these scenarios, the algorithm will work as follows:

System Initialization: Controllers will be assigned roles according to their GUID. The Primary is assigned to the lowest GUID, Secondary to the second lowest, and Tertiary to the highest GUID.

While the system is designed to work with three controllers, it should work independently of Fault Tolerance with one or two controllers. Any late joining controllers always come in at the next available highest role. I.e., A single controller as Primary, with subsequent controllers coming in as Secondary and Tertiary. Two controllers coming up at the same time (within 10 seconds of each other) will come up as Primary and Secondary based on GUID value.

To allow "Initialization Time" as controllers come up within ten seconds of each other, they will wait for either ten seconds from the last controller that was recognized (via Heartbeat messages) or three controllers. Once the "Initialization Time" is concluded, each controller will cast a vote for Primary, Secondary, and Tertiary. The controller with the greatest number of votes for each roll will be assigned that roll. Note: All controllers should have a common view, so all voting should be unanimous for each roll. In the event of a non-unanimous vote, the source GUID of the minority vote should be marked FAILED.

Late Joiner: To avoid Oscillation, a controller, once a Primary, should remain in that role as other controllers come and go unless it is the controller that fails. The same is true of the Secondary if there is a Primary; that is, any third controller will come in as Tertiary. Should a controller fail, other controllers will be 'Promoted' to the next highest roll (e.g., A Primary failure causes both the Secondary and Tertiary to each be promoted to Primary and Secondary, and a Secondary failure will cause the Tertiary to be promoted to Secondary. Late Joiners always come in at the lowest available role.

A controller is able to make a distinction as to which scenario it is in based on receiving a vote prior to reaching the Voting State. Votes are durable, so receiving a vote prior indicates the control is a Late Joiner to a running system. Receiving votes after the Vote state indicates a controller is in the System Initialization scenario. NOTE: To be deterministic, all controllers must either wait until they see three unique heartbeats (i.e., three controllers) or ten seconds after the last unique heartbeat. This ensures all controllers depart the "Initialization Time" within 100ms of each other (and will not receive another controller's vote in this phase unless it is a Late Joiner.)

No Pixy/Servo topics should be written until a Primary has been selected and a controller has established its own role.

Additional controllers beyond three will be ignored.

Voted Result (non-topic):

Depending on how paranoid we are, a separate Vote Result announcement topic could be published durably by the Primary Controller. However, this design will allow the durable Voting topic to suffice. Any unit that does not agree with the two votes of the other controllers will take itself offline and/or restart. This ensures that the Vote Topic essentially reflects any voting result topic.

ReVoting - Revoting will occur anytime a tracker is removed or added (or comes back into operation). Ideally, we'd trigger on a deadline miss and identify the lost writer (Tracker). We need the instance of the HB writer that missed its deadline so we can clear out the database (RedundancyInfo) for that writer and revote. Instead, we'll run the SM at 1 second and the HB rate at 250ms. A deadman count will be employed. Each HB will increment an internal count (kept in the RedundancyInfo struct.) When the state machine is in STEADY_STATE, it will check the count for non-zero and then clear this count. The corresponding Tracker will be nulled out of the RedundancyInfo struct, and the SM will revert to the VOTE state. During this time, if the primary had failed, the secondary will be driving the servo topic (the servo topic has ownership strength with a 100ms deadline)

LED Operation:
Each Controller will have four red/green LEDs.

They will appear:

ROLE			STATUS
LED1	LED2	LED3	LED4
Primary	Secondary	Tertiary	OK/Fail

Each Tracker working tracker will present a Green LED indicating its role.
A Red LED will indicate a failed unit and role. Off indicates a Tracker with a corresponding back up role is Operational.

LED4 will indicate the Trackers Own status and is redundant with the ROLE Status but helpful upon initialization prior to roles being assigned and if the trackers software detects its own failure.

Note: corresponding messages will be printed on the controller console indicating each controllers Relative Number, and who is primary.

PixyTracker Code Modifications

Expected Behavior

Upon startup, a system will wait a 10-second grace period to allow all controllers to boot up. Depending upon the number of detected controllers, enter one of the described three states above and be deemed operational. During this operational phase, the system may degrade to a lower redundancy state or upgrade to a higher redundancy state sending the appropriate alarms and informational messages as described previously. It is up to the operator to take action to minimize MTTR and maintain the highest availability.

All active controllers send intended Command topics regularly whether or not Circle topics arrive, and pixy/servo_control commands are needed.

At least one controller should be sending pixy/servo_updates and possibly two or all three. In the latter cases, the units should not be in simplex mode, but all three should agree they are in 3-way voting mode. In either case, only the pixy/servo_control topic with the highest ownership will be sent to the pixy cam/ pixyshapes app by DDS. Each controller always sends a pixy/servo_control topic at the fixed unique ownership strength it is assigned at startup.

Upon switchover, the 'matching' pixy/servo_control commands may differ by an acceptable error factor (see Value accuracy match above). Thus, the pixy servo may have a slight jitter movement between switchovers and then track per normal operation.

DDS Mechanisms Discussed:

DDS Mechanism	Component Used*	Description
Liveliness	Controller Components	Detects a redundant component / participant failure
Ownership	Between Controller and Actuator	Enables the receiving application, independent of the redundancy strategy used to see at most two samples (one on each bus A and B)
Key'd data	Controller Components	Enables redundant components to use liveliness to detect each controller in a 3-way voting system
Match on Publisher	Controller Components	Enables the system to determine how many controllers are active.
Durability**	Controller Components	Allows a newly joined controller in a system that calculates the value via integrating a number of samples to more quickly get within the acceptable value error more quickly (i.e. become a fully redundant partner)

* Shown only for a simple example where Controller only publishes and Actuator only subscribes. In a closed loop system add the Actuator Component or 'Between Actuator and Controller'

** Durability is not used in the FAE project implementation, but where data is integrated to get result, a newly joined controller would require a grace period of n-samples to get enough integration to be 'in spec'. Using durability could shorten this grace period (i.e. amount of time to be fully redundant)

Interesting Observations

DDS Ownership decouples Switchover Time from the (re)voting process.

Using DDS Ownership, QoS decouples the "Switchover Time" from the time to revote and elect a new Primary. When the Primary fails, after the Pixy/Servo topic deadline is missed, the Secondary's sample will be sent to the application. Voting can then go on in the background to promote the Secondary (and associated writer strength) to the Primary.

Exclusive Ownership on aperiodic topics -

Topics that are not periodic and have Exclusive Ownership, such as the pixy/Servo topic, behave interestingly. Exclusive Ownership uses a deadline expiration for DDS to send the next highest strength writer's sample to the application. Aperiodic topics may routinely miss deadlines and could cause DDS to act on a sample from a lower-strength writer. After the deadline is missed, if all writers of the topic send samples in unison and the lower strength sample arrives first, that sample will be passed up to the application. Because the higher-strength writer is also writing, once the higher-strength sample is received, DDS will send only higher-strength samples until the deadline is again missed. There are three solutions to this issue: 1) Do nothing. This is applicable IFF an extra sample with similar or the same data will not cause a problem. 2) Make the topic periodic. If data is not changing, send the last sample before the deadline expires. 3) Don't use Excessive Ownership. Instead, allow only the Primary writer to send samples (i.e., one writer at a time. This impacts switchover time as it is now dependent upon the time necessary to recognize a missing heartbeat (or two), revote, evaluate the results, and enable the new Primary Controller.

Dev Plan:

Development can be done on a host PC will three different Tracker apps running, prior to being moved the each Raspberry Pi

1. +Put 64-Bit Debian Linux (Bullseye) on RPI 4
2. +Put 64-Bit Debian Linux (Bullseye) and DDS 7.2. on RPI3s - 3 of them
3. +Recompile PixyShapes (With Pixy Cam) on RPI 4 with DDS 7.2
4. +Add ownership to servo-control topic - show it works
5. +Get GPIO package running on RPI3s to provide LED status
6. +Convert PIxyTracker to C++11 using multitopic code factoring wrappers (on mac)
7. + Add in Heartbeat and voting topics
8. + Add Voting algorithm and change ownership on servo-control topic accordingly
9. - Add LED update main thread
10. - Move to RPI3s