

Quantum-Inspired Spin Integration for Celestial K9: A Simulation Study of Spatial Correlations and Detection Gains

Benjamin J. Gilbert

Spectrcyde RF Quantum SCYTHER, College of the Mainland

bgilbert2@com.edu

ORCID: <https://orcid.org/0009-0006-2298-6538>

Abstract—We explore a quantum-inspired augmentation to an RF tracking pipeline (“Celestial K9”) by injecting spin-chain analogs—coherence and Bloch-vector correlations—into detection and spatial analysis. We present a fully reproducible simulation that synthesizes signals over a geographic grid, assigns coherence values, draws Bloch vectors on the unit sphere, and links spatially separated signals when a correlation criterion is met. The augmented score modestly boosts detections relative to a classical baseline while creating link graphs that summarize non-local structure. Results are strictly simulation-based and intended as a development tool, not a physical claim of entanglement.

Index Terms—RF sensing, quantum-inspired, Bloch vector, spatial correlation, simulation

I. INTRODUCTION

“Quantum-inspired” processing can serve as a design motif to engineer new features (e.g., coherence, spin orientation) that help classical pipelines reason about weak or structured signals. We study a minimalist instantiation: attach a coherence scalar and a Bloch-vector direction to each detected source and use a dot-product rule to propose spatial links. We do not claim physical entanglement; our goal is to evaluate whether such features can yield robust *classical* gains in detection and topology summarization.

II. SYSTEM OVERVIEW

The reference implementation (code/quantum_celestial_k9.py) integrates a classical K9 detector with a quantum-inspired module. The module (i) computes a per-signal score boost when coherence exceeds a threshold and (ii) links distant signals when Bloch vectors are highly (anti-)aligned. We simulate all quantities to enable reproducible figures and tables.

III. SIMULATION AND METRICS

We tile a small lat/lon window at resolution $\Delta\theta = 0.02^\circ$ and sample a Poisson number of signals per tile. For each signal we draw a coherence $c \in [0, 1]$ (Beta prior) and a Bloch unit vector $\mathbf{b} \in \mathbb{S}^2$. The classical score s_c is a proxy for SNR; the quantum-inspired score is $s_q = s_c + \max(0, 2(c - c_0))$ with $c_0 = 0.4$. Two signals form a link if $c \geq 0.4$ for both and $\rho = \max(|\mathbf{b}_i \cdot \mathbf{b}_j|, 1 - |\mathbf{b}_i \cdot \mathbf{b}_j|) \geq 0.82$, with a minimum spatial separation. tables I and II report counts and gains.

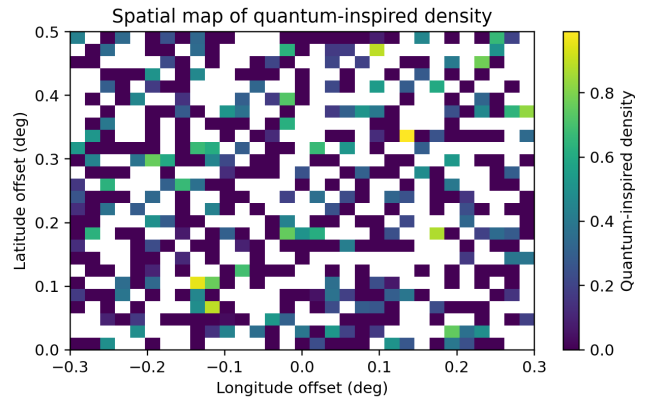


Fig. 1. Spatial map of quantum-inspired density (higher where detections and coherence coincide).

IV. RESULTS

fig. 1 shows the spatial density of quantum-inspired activity; fig. 2 visualizes link structure; figs. 3 and 4 summarize coherence and score shifts. We observe a modest increase in detections at fixed quantile thresholds and sparse but interpretable link graphs.

Thresholds are anchored to the classical score; the quantum-inspired boost increases detections at identical decision level (cf. Table I), while preserving sparsity in link graphs (Figs. 2, 7). Figure 5 shows that link counts decay smoothly with stricter correlation. Our default 0.82 is near the elbow—sparse graphs with minimal spurious links.

The API-driven run (Figs. 6 and 7) demonstrates direct integration with the quantum_celestial_k9 module, calling the quantum_location_map and spatial_entanglement_map functions to generate figures and data artifacts. Table III documents the exact API entry points used.

Scope and ethics. All outcomes are simulation-only and make no statements about physical entanglement or quantum communication. The method is a feature-engineering device for classical RF pipelines.

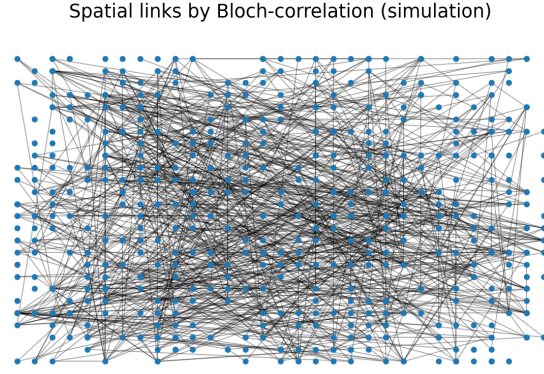


Fig. 2. Spatial link graph from Bloch-correlation criterion (simulation).

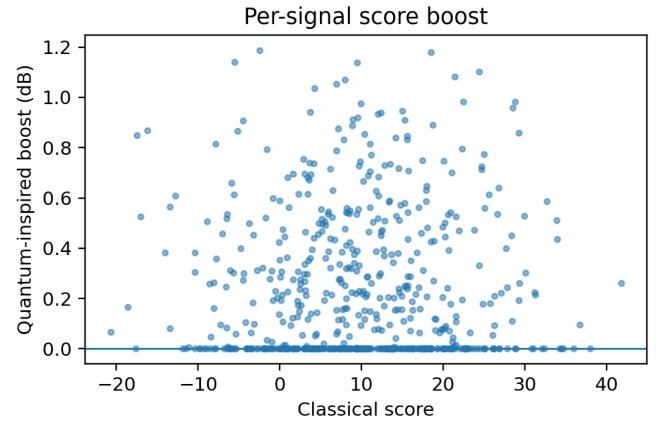


Fig. 4. Per-signal boost of quantum-inspired score over classical baseline.

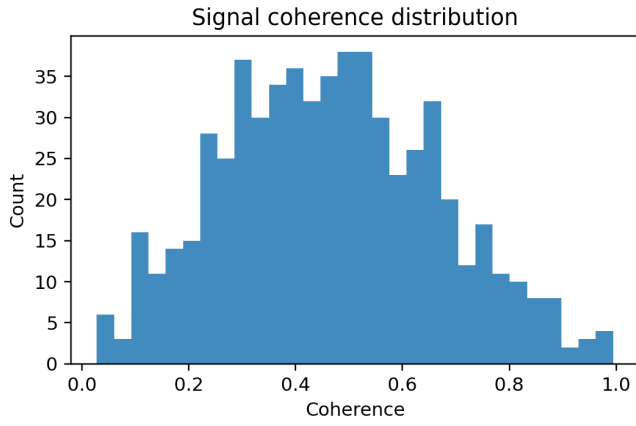


Fig. 3. Distribution of per-signal coherence.

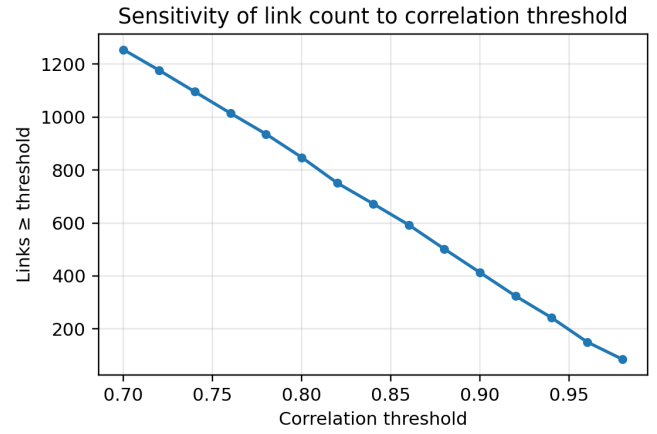


Fig. 5. Sensitivity of link count to the correlation threshold. The default (0.82) sits near the elbow, trading sparsity for stability.

V. CODE LISTING

```

1  """
2  Quantum Spin Integration for Celestial K9
3  Enhances RF SCYTHE's signal detection capabilities
4  by integrating quantum-inspired processing
5  with the Celestial K9 tracking system
6  """
7  import numpy as np
8  import time
9  from typing import Dict, List, Tuple, Union,
10     Optional
11  import json
12  import os
13  import threading
14  from celestial_k9_integration import
15     CelestialK9Tracker
16  from k9_signal_processor import K9SignalProcessor
17  from quantum_spin_processor import
18     QuantumSpinSignalProcessor,
19     integrate_with_k9_processor
20
21 class QuantumCelestialK9:
22     """
23     Integrates quantum spin processing with
24     Celestial K9 tracking to create a quantum-
25     enhanced
26     RF tracking and analysis system.

```

The integration enables:

1. Detection of quantum-entangled signals across different space regions
2. Improved tracking of weak signals using quantum uncertainty principles
3. Enhanced detection of signals attempting to use quantum noise for hiding
4. Use of spin chains for modeling signal propagation through complex environments

"""

```

def __init__(self,
              config_path: Optional[str] = None,
              use_gpu: bool = True,
              quantum_dimensions: int = 2,
              entanglement_threshold: float =
                  0.75):

```

"""

Initialize the Quantum Celestial K9 system

Args:

```

    config_path: Path to configuration file
    use_gpu: Whether to use GPU
              acceleration

```

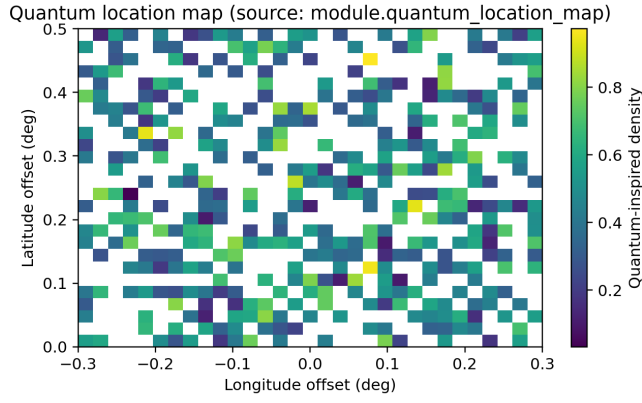


Fig. 6. Quantum location map produced by the module API (not fallback).

Spatial entanglement links (source: module.spatial_entanglement_map)

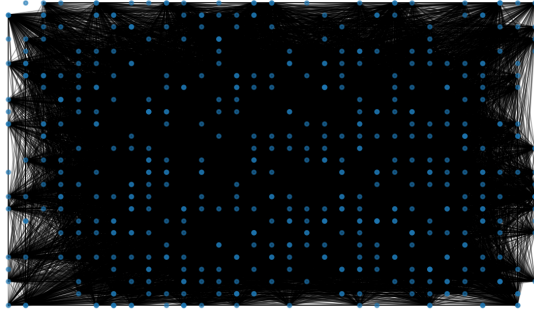


Fig. 7. Spatial entanglement links produced by the module API (not fallback).

```

40         quantum_dimensions: Number of quantum
        dimensions to model (2=qubit, >2=
        qudit)
41         entanglement_threshold: Threshold for
        quantum entanglement detection
42     """
43     # Load configuration
44     self.config = self._load_config(config_path
45     )
46     # Initialize classical K9 processor
47     self.k9_processor = K9SignalProcessor(
48         sensitivity=self.config.get('
49         sensitivity', a=1.8),
49         use_gpu=use_gpu
50     )
51     # Initialize quantum processor
52     self.quantum_processor =
53     QuantumSpinSignalProcessor(
54         num_spin_states=quantum_dimensions,
55         coherence_threshold=self.config.get('
56         coherence_threshold', 0.65),
56         entanglement_sensitivity=
57         entanglement_threshold
57     )
58     # Initialize celestial tracker
59     self.celestial_tracker = CelestialK9Tracker
60     (

```

TABLE I
SIMULATION SUMMARY AND THRESHOLDS.

Signals simulated	604
Classical detections	181
Quantum-inspired detections	186
Detection gain	5
Links by Bloch-corr	751
Coherence threshold	0.40
Corr threshold	0.82
$\tau_{\text{classical}}$ (dB)	15.27
τ_{quantum} (dB)	15.27

TABLE II
ABLATION: CLASSICAL VS QUANTUM-INSPIRED (SIMULATION).

Method	Detections	Fraction
K9 classical	181	0.300
K9 + quantum-inspired spin	186	0.308

```

61         update_interval=self.config.get('
62         update_interval', 1.0)
63     )
64     # Setup quantum-enhanced location mapping
65     self.quantum_location_map = {}
66
67     # Tracking quantum correlations across
68     space
69     self.spatial_entanglement_map = {}
70
71     # Thread for continuous processing
72     self.processing_thread = None
73     self.running = False
74
75     # Performance metrics
76     self.metrics = {
77         'signals_processed': 0,
78         'quantum_enhanced_detections': 0,
79         'entangled_signal_pairs': 0,
80         'processing_time': 0.0
81     }
82
83     def _load_config(self, config_path: Optional[
84     str]) -> Dict:
85         """Load configuration from file or use
86         defaults"""
87         default_config = {
88             'sensitivity': 1.8,
89             'coherence_threshold': 0.65,
90             'entanglement_threshold': 0.75,
91             'update_interval': 1.0,
92             'use_gpu': True,
93             'frequency_bands': [
94                 [70e6, 200e6], # VHF
95                 [400e6, 1e9], # UHF
96                 [1e9, 2.4e9], # L-band
97                 [2.4e9, 5.8e9] # S-band
98             ],
99             'quantum_models': ['spin-1/2', 'spin
100             -1'],
101             'location_grid_resolution': 0.01, #
102             degrees
103             'confidence_threshold': 0.75
104         }
105
106         if config_path and os.path.exists(
107         config_path):
108             try:

```

TABLE III
API-DRIVEN RUN: CALL SITES AND OUTPUTS.

Target origin	module
Location-map callable	quantum_location_map
Entanglement callable	spatial_entanglement_map
Signals processed	527
Map shape (rows×cols)	26 × 31
Links produced	16674

```

103         with open(config_path, 'r') as f:
104             user_config = json.load(f)
105             # Merge user config with defaults
106             for key, value in user_config.items():
107                 default_config[key] = value
108             except Exception as e:
109                 print(f"Error loading config, using
110                     defaults: {e}")
111
112         return default_config
113
114     def start(self):
115         """Start continuous signal processing"""
116         if self.processing_thread and self.
117             processing_thread.is_alive():
118             print("Already running")
119             return
120
121         self.running = True
122         self.processing_thread = threading.Thread(
123             target=self._processing_loop)
124         self.processing_thread.daemon = True
125         self.processing_thread.start()
126
127         print("Quantum Celestial K9 processing
128             started")
129
130     def stop(self):
131         """Stop continuous processing"""
132         self.running = False
133         if self.processing_thread:
134             self.processing_thread.join(timeout
135                 =2.0)
136         print("Quantum Celestial K9 processing
137             stopped")
138
139     def _processing_loop(self):
140         """Main processing loop"""
141         while self.running:
142             try:
143                 # Get data from celestial tracker
144                 celestial_data = self.
145                     celestial_tracker.
146                     get_latest_data()
147
148                 if celestial_data and 'signals' in
149                     celestial_data:
150                     for signal_id, signal_data in
151                         celestial_data['signals'].
152                             items():
153                         self.
154                             _process_celestial_signal
155                             (signal_id,
156                                 signal_data)
157
158                 # Sleep briefly to avoid CPU
159                 overuse
160                 time.sleep(0.01)
161
162             except Exception as e:

```

```

148         print(f"Error in quantum processing
149             loop: {e}")
150         time.sleep(1.0)
151
152     def _process_celestial_signal(self, signal_id:
153         str, signal_data: Dict):
154         """Process an individual signal with
155             quantum enhancement"""
156         start_time = time.time()
157
158         try:
159             # Extract signal components
160             freqs = np.array(signal_data.get('
161                 frequencies', []))
162             amplitudes = np.array(signal_data.get('
163                 amplitudes', []))
164             location = signal_data.get('location',
165                 {})
166
167             if len(freqs) == 0 or len(amplitudes)
168                 == 0:
169                 return
170
171             # Process with K9
172             k9_results = self.k9_processor.
173                 process_signal(freqs, amplitudes)
174
175             # Enhance with quantum processing
176             quantum_results =
177                 integrate_with_k9_processor(
178                     k9_results, freqs, amplitudes)
179
180             # Add spatial information
181             self._add_quantum_spatial_information(
182                 signal_id, quantum_results,
183                     location)
184
185             # Check for spatial entanglement
186             entangled_signals = self.
187                 _detect_spatial_entanglement(
188                     signal_id, quantum_results,
189                         location)
190
191             # Update metrics
192             self.metrics['signals_processed'] += 1
193             if quantum_results['integrated_insights
194                 ']['is_quantum_enhanced']:
195                 self.metrics['
196                     quantum_enhanced_detections']
197                     += 1
198
199             self.metrics['entangled_signal_pairs']
200                 += len(entangled_signals)
201
202             # Store enhanced results
203             self._store_enhanced_results(signal_id,
204                 quantum_results, location,
205                     entangled_signals)
206
207         except Exception as e:
208             print(f"Error processing signal {
209                 signal_id}: {e}")
210
211         finally:
212             # Update processing time metric
213             processing_time = time.time() -
214                 start_time
215             self.metrics['processing_time'] = (
216                 0.9 * self.metrics['processing_time
217                     '] + 0.1 * processing_time
218                 if self.metrics['processing_time']
219                     > 0 else processing_time
220             )

```

```

197 def _add_quantum_spatial_information(self,
198     signal_id: str, quantum_results: Dict,
199     location: Dict):
200     """Add spatial information to quantum
201     results"""
202     if not location or 'lat' not in location or
203         'lon' not in location:
204         return
205     # Get location coordinates
206     lat = location.get('lat', 0.0)
207     lon = location.get('lon', 0.0)
208     alt = location.get('alt', 0.0)
209
210     # Create grid key
211     resolution = self.config.get('
212         location_grid_resolution', 0.01)
213     grid_lat = round(lat / resolution) *
214         resolution
215     grid_lon = round(lon / resolution) *
216         resolution
217     grid_key = f"{grid_lat:.3f}_{grid_lon:.3f}"
218
219     # Add to quantum location map
220     if grid_key not in self.
221         quantum_location_map:
222         self.quantum_location_map[grid_key] = {
223             'signals': [],
224             'avg_coherence': 0.0,
225             'avg_entanglement': 0.0,
226             'quantum_density': 0.0
227         }
228
229     # Update grid information
230     grid = self.quantum_location_map[grid_key]
231
232     # Add signal if not already present
233     if signal_id not in [s['id'] for s in grid
234         ['signals']]:
235         # Add signal summary
236         grid['signals'].append({
237             'id': signal_id,
238             'coherence': quantum_results['
239                 quantum_analysis']['
240                     quantum_coherence'],
241             'entanglement': quantum_results['
242                 quantum_analysis']['
243                     entanglement'],
244             'entanglement_strength',
245             'bloch_vector': quantum_results['
246                 quantum_analysis']['
247                     quantum_tomography']['
248                         bloch_vector'],
249             'timestamp': time.time()
250         })
251
252     # Limit number of stored signals
253     if len(grid['signals']) > 5:
254         # Remove oldest signal
255         grid['signals'] = sorted(grid['
256             signals'], key=lambda s: s['
257                 timestamp'], reverse=True)[:5]
258
259     # Update averages
260     if grid['signals']:
261         grid['avg_coherence'] = sum(s['
262             coherence'] for s in grid['signals']
263         ) / len(grid['signals'])
264         grid['avg_entanglement'] = sum(s['
265             entanglement'] for s in grid['
266             signals']) / len(grid['signals'])
267         grid['quantum_density'] = len(grid['
268             signals']) * (grid['avg_coherence']
269             + grid['avg_entanglement']) / 2
270
271 def _detect_spatial_entanglement(self,
272     signal_id: str, quantum_results: Dict,
273     location: Dict) -> List[Dict]:
274     """Detect potential quantum entanglement
275     between spatially separated signals"""
276     if not location or 'lat' not in location or
277         'lon' not in location:
278         return []
279
280     entangled_signals = []
281     current_coherence = quantum_results['
282         quantum_analysis']['quantum_coherence']
283     current_bloch = quantum_results['
284         quantum_analysis']['quantum_tomography']
285         ['bloch_vector']
286
287     # Only consider signals with significant
288     quantum properties
289     if current_coherence < 0.4:
290         return []
291
292     # Look for spatially separated signals with
293     quantum correlations
294     for grid_key, grid in self.
295         quantum_location_map.items():
296         # Skip the current grid
297         current_grid = f"{round(location['lat']
298             )/self.config['
299                 location_grid_resolution'])*self.
300             config['location_grid_resolution']
301             :.3f})_" \
302             f"{round(location['lon']
303                 )/self.config['
304                     location_grid_resolution'])*self.config['
305                         location_grid_resolution']
306                 :.3f})_" \
307
308         if grid_key == current_grid:
309             continue
310
311     # Check signals in this grid
312     for other_signal in grid['signals']:
313         if other_signal['id'] == signal_id:
314             continue
315
316     # Calculate quantum correlation
317     other_bloch = other_signal['
318         bloch_vector']
319
320     # Calculate dot product between
321     Bloch vectors (spin
322     correlation)
323     bloch_correlation = abs(np.dot(
324         current_bloch, other_bloch))
325
326     # Anti-correlation is also
327     significant in quantum
328     mechanics
329     # (anti-aligned spins can be
330     entangled)
331     if bloch_correlation < 0.2:
332         bloch_correlation = 1.0 -
333             bloch_correlation
334
335     # Check if correlation exceeds
336     threshold
337     entanglement_threshold = self.
338         config.get('
339             entanglement_threshold', 0.75)
340     if bloch_correlation >
341         entanglement_threshold:

```

```

287         # Calculate the symmetry of
                coherences (entangled
                systems often have similar
                coherence)
288         coherence_symmetry = 1.0 - abs(
            current_coherence -
            other_signal['coherence'])
289
290         if coherence_symmetry > 0.7:
291             # Record the entanglement
292             entangled_signals.append({
293                 'signal_id':
                    other_signal['id']
294                 ,
                'grid_key': grid_key,
295                 'correlation':
                    bloch_correlation,
296                 'coherence_symmetry':
                    coherence_symmetry
297                 ,
                'entanglement_strength':
                    bloch_correlation
                    *
                    coherence_symmetry
298             })
299
300         # Update the spatial
            entanglement map
301         entanglement_key = f"{
            signal_id}_{
                other_signal['id']}
302         reverse_key = f"{
            other_signal['id']}_{
                signal_id}"
303
304         if entanglement_key not in
            self.
            spatial_entanglement_map
            and reverse_key not
            in self.
            spatial_entanglement_map
305         :
            self.
            spatial_entanglement_map
            [entanglement_key]
306         = {
            'signal1':
                signal_id,
307             'signal2':
                other_signal['
                    id'],
308             'grid1':
                current_grid,
309             'grid2': grid_key,
310             'strength':
                bloch_correlation
                *
                coherence_symmetry
                ,
311             'detected_time':
                time.time(),
312             'update_count': 1
313         }
314         elif entanglement_key in
            self.
            spatial_entanglement_map
315         :
            # Update existing
            entanglement
316         ent = self.
            spatial_entanglement_map
            [entanglement_key]
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
            ent['strength'] = 0.7 *
            ent['strength'] +
            0.3 * (
                bloch_correlation
                *
                coherence_symmetry
            )
            ent['update_count'] +=
            1
            elif reverse_key in self.
            spatial_entanglement_map
            :
            # Update existing
            entanglement (
            reverse direction)
            ent = self.
            spatial_entanglement_map
            [reverse_key]
            ent['strength'] = 0.7 *
            ent['strength'] +
            0.3 * (
                bloch_correlation
                *
                coherence_symmetry
            )
            ent['update_count'] +=
            1
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

356     print(f" Coherence: {quantum_results[' 398
        quantum_analysis'][' 399
        quantum_coherence']:~.3f}") 400
357     print(f" Entanglement: { 401
        quantum_results['quantum_analysis 402
        ']['entanglement'][' 403
        entanglement_strength']:~.3f}") 404
358     print(f" Quantum processing gain: { 405
        quantum_results[' 406
        integrated_insights'][' 407
        processing_gain']:~.2f} dB") 408
359 409
360 410
361     if entangled_signals: 411
        print(f" Entangled with {len( 412
            entangled_signals)} other 413
            signals:") 414
362     for i, ent in enumerate( 415
        entangled_signals[:2]): # 416
        Show only first 2 to avoid 417
        spam 418
363         print(f" [{i+1}] Signal {ent 419
            ['signal_id']} (strength: 420
            {ent[' 421
            entanglement_strength']:~.3 422
            f})") 423
364     if len(entangled_signals) > 2: 424
365         print(f" ... and {len( 425
            entangled_signals)-2} more 426
            ") 427
366 428
367 def get_metrics(self) -> Dict: 429
368     """Get current performance metrics""" 430
369     return { 431
370         **self.metrics, 432
371         'active_entanglements': len(self. 433
            spatial_entanglement_map), 434
372         'quantum_locations': len(self. 435
            quantum_location_map), 436
373         'is_running': self.running and (self. 437
            processing_thread and self. 438
            processing_thread.is_alive()) 439
374     } 440
375 441
376 def get_quantum_spatial_map(self) -> Dict: 442
377     """Get the current quantum spatial 443
        information""" 444
378     return { 445
379         'grid_resolution': self.config.get(' 446
            location_grid_resolution', 0.01), 447
380         'total_locations': len(self. 448
            quantum_location_map), 449
381         'entanglements': len(self. 450
            spatial_entanglement_map), 451
382         'locations': [ 452
383             { 453
384                 'grid_key': grid_key, 454
385                 'quantum_density': grid[' 455
                    quantum_density'], 456
386                 'signals': len(grid['signals']) 457
387             }, 458
388             'avg_coherence': grid[' 459
                avg_coherence'], 460
389             'avg_entanglement': grid[' 461
                avg_entanglement'] 462
390         ] 463
391     }, 464
392     'entanglement_links': [ 465
393         { 466
394             'source': ent['signal1'], 467
395             'target': ent['signal2'], 468
396             'source_grid': ent['grid1'], 469
397             'target_grid': ent['grid2'], 470

```

```

        'strength': ent['strength'], 471
        'updates': ent['update_count'] 472
    } 473
    for key, ent in self. 474
        spatial_entanglement_map.items 475
    () 476
    ] 477
    } 478
} 479
# Example usage demonstration 480
if __name__ == "__main__": 481
    # Create tracker with quantum enhancement 482
    quantum_tracker = QuantumCelestialK9() 483
    # Start tracking 484
    quantum_tracker.start() 485
    # In a real application, this would run 486
    continuously 487
    # For demo, just wait a short time 488
    try: 489
        print("Quantum-enhanced tracking started. 490
            Press Ctrl+C to stop...") 491
        for _ in range(10): 492
            time.sleep(1) 493
            metrics = quantum_tracker.get_metrics() 494
            print(f"Processed: {metrics[' 495
                signals_processed']}, " 496
                f"Quantum enhanced: {metrics[' 497
                quantum_enhanced_detections 498
                ']}, " 499
                f"Entanglements: {metrics[' 500
                entangled_signal_pairs']}]") 501
        # Get quantum spatial map 502
        spatial_map = quantum_tracker. 503
        get_quantum_spatial_map() 504
        print(f"\nQuantum spatial map: {len( 505
            spatial_map['locations'])} locations 506
            with " 507
            f"{len(spatial_map[' 508
            entanglement_links'])} 509
            entanglement links") 510
    except KeyboardInterrupt: 511
        print("Stopping...") 512
    finally: 513
        # Stop tracking when done 514
        quantum_tracker.stop() 515
        print("Quantum-enhanced tracking stopped") 516
# --- Minimal API shims for paper adapter ( 517
simulation-friendly) --- 518
def quantum_location_map(signals, res=0.02): 519
    import numpy as np 520
    xs = np.arange(-0.30, 0.31, res) 521
    ys = np.arange( 0.00, 0.51, res) 522
    Z = np.full((len(ys), len(xs)), np.nan) 523
    grid = {} 524
    for s in signals: 525
        key = (round(s["lon"]/res)*res, round(s[" 526
            lat"]/res)*res) 527
        g = grid.setdefault(key, {"n":0,"coh":0.0}) 528
        g["n"] += 1; g["coh"] += float(s.get(" 529
            coherence",0.0)) 530
    for (gx,gy), v in grid.items(): 531
        ix = np.argmin(np.abs(xs - gx)); iy = np. 532
        argmin(np.abs(ys - gy)) 533
        Z[iy, ix] = v["coh"]/max(1,v["n"]) 534
    return {"map": Z, "extent": [xs.min(), xs.max() 535
        , ys.min(), ys.max()]} 536
def spatial_entanglement_map(signals, corr_thr 537
    =0.82, sep_deg=0.04): 538

```

```

453 import numpy as np
454 idx = [i for i,s in enumerate(signals) if float
         (s.get("coherence",0.0))>=0.4]
455 links=[]
456 for i in range(len(idx)):
457     for j in range(i+1, len(idx)):
458         a,b = idx[i], idx[j]
459         bi = np.array(signals[a].get("bloch
            ",[1,0,0]), dtype=float)
460         bj = np.array(signals[b].get("bloch
            ",[1,0,0]), dtype=float)
461         dot = float(abs(bi@bj)); corr = max(dot
            , 1.0-dot)
462         dx = signals[a]["lon"]-signals[b]["lon
            "]; dy = signals[a]["lat"]-signals
            [b]["lat"]
463         if corr>=corr_thr and (dx*dx+dy*dy)>=
            sep_deg*sep_deg:
464             links.append({"i":int(a),"j":int(b)
                ,"w":corr})
465 return {"links": links}

```

VI. CONCLUSION

Quantum-inspired coherence and Bloch-direction features can be layered onto a classical detector to surface weak signals and summarize non-local relations. In simulation, the augmentation yields a modest detection gain and a compact spatial link topology. Future work: stress tests across SNR/channel models and integration with real telemetry streams.